Towards Robust Long-form Text Generation Systems

Kalpesh Krishna
University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/dissertations_2

Part of the Artificial Intelligence and Robotics Commons, Data Science Commons, and the Information Security Commons

Recommended Citation
https://doi.org/10.7275/35984373 https://scholarworks.umass.edu/dissertations_2/3004

This Open Access Dissertation is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Doctoral Dissertations by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.
TOWARDS ROBUST LONG-FORM TEXT GENERATION SYSTEMS

A Dissertation Presented
by
KALPESH KRISHNA

Submitted to the Graduate School of the University of Massachusetts Amherst in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2023

Robert and Donna Manning College of Information and Computer Sciences
TOWARDS ROBUST LONG-FORM TEXT GENERATION SYSTEMS

A Dissertation Presented

by

KALPESH KRISHNA

Approved as to style and content by:

__________________________________________
Mohit Iyyer, Chair

__________________________________________
Andrew McCallum, Member

__________________________________________
Hamed Zamani, Member

__________________________________________
Yejin Choi, Member

__________________________________________
Ramesh K. Sitaraman, Associate Dean for
Educational Programs and Teaching
Robert and Donna Manning College of
Information and Computer Sciences
ACKNOWLEDGMENTS

First and foremost, I would like to thank my advisor Mohit Iyyer, for all his guidance, support and encouragement throughout the five years. Right from the start, Mohit encouraged me to work on ambitious research projects, while giving me the freedom to explore research ideas I found most interesting. Our weekly 1:1s were always the highlight of my week, and I am happy we often discussed ideas beyond the immediate research project, which sowed the seeds for future research endeavours! Throughout the PhD, Mohit went the extra mile to ensure I was productive and happy. In my very first conference submission, he stayed up with me till 2AM in the lab during a freezing Massachusetts winter night, to ensure we submitted a high quality draft by the deadline. During the unprecedented COVID-19 pandemic, he would regularly check in with me, and provided me with work-from-home equipment as well as special lab access to ensure I could stay productive. Finally during the stressful job search, he made sure I was well-prepared, and regularly provided me with career perspectives to help my decision making. I am also grateful for his encouragement to do industrial internships in every PhD summer, which enabled me to pursue ambitious ideas requiring large-scale resources. Mohit has helped me achieve well beyond what I thought was possible, and I am indebted to him for helping me blossom into a mature researcher.

I would like to thank the other members of my committee, Andrew McCallum, Hamed Zamani and Yejin Choi for several insightful discussions which helped me improve this document, and my job talk. I am especially thankful to Andrew for encouraging me to join UMass back in 2018, and for career advice on various occasions. I have been very lucky to pursue my PhD alongside wonderful labmates, from whom I have learned so much! I’ll never forget our lab retreats, bike rides, hikes, and late night conversations, which made my time at UMass so much fun! In particular, I’d like to thank Nader Akoury,
Marzena Karpinska, Tu Vu, Justin Payan, Javier Burroni, Rajarshi Das, Katie Keith, Abe Handler, Su Lin Blodgett, Andrew Drozdov, Xiang Lorraine Li, Neha Kennard, Simeng Sun, Shufan Wang, Katherine Thai, Yapei Chang, Dongxu Zhang, Ankita Gupta, Chau Pham, Emma Strubell, Pat Verga, Nick Monath and Subendhu Rongali for all the fun times! I am grateful to Brendan O’Connor and other UMass CICS faculty for their support and guidance throughout my PhD. Finally, a big thank you to the tireless UMass CICS staff, especially Eileen Hamel for helping me navigate numerous logistic issues during my PhD.

Internships were one of the highlights of my PhD, and I would like to thank all my supervisors for hosting me and encouraging me to do good research: Gaurav Singh Tomar, Ankur Parikh, Aurko Roy, Partha Talukdar, Bidisha Samanta, John Wieting, Kyle Lo, Arman Cohan, and Pradeep Dasigi. John deserves a special mention, for being a close collaborator throughout my PhD, and helping shape a number of chapters of this thesis. I am also grateful to my co-interns at Google and Allen AI—especially Nitish Gupta, John Giorgi, Lucy Li, Rajarshi Das, Carl Edwards, Shaurya Rohatgi, and Hila Gonen—for the fun times we had together. The numerous games of 8-ball pool, jigsaw puzzles, and the 2AM hike to Sunrise in Mount Rainier! Of course, we worked very hard on our projects too!

One of the biggest positives for me living in Amherst has been learning to appreciate the natural beauty all around us. During the COVID-19 pandemic, I started bird watching, a hobby that quickly evolved into a passion, thanks to the wonderful birding community in the Pioneer Valley. A big thank you to Steve Winn and Lesley Farlow, for all the wonderful birding walks which always resulted in “lifers”, and to Greg Brown, Joe Oliverio, Mary McKittrick, Larry Therrien, Cory Elowe, the Hampshire Bird Club, and the UMass Bird Club. I promise I will continue my quest to find the elusive Connecticut Warblers! I would also like to thank all my roommates and Amherst friends for making my time off work entertaining. In particular, I thank Bhanu Teja Gullapalli, Forsad Al Hossain, Vaishali Malik, Farhan and Farhana Oshim, Manasa Kalanadhabhatta, and Tejas Savalia. I will never forget all our birthday celebrations, competitive board games and badminton matches.
Thank you to my mentors during the undergraduate days at IIT Bombay—Preethi Jyothi, Karen Livescu, Kevin Gimpel, Shubham Toshniwal and Liang Lu, for providing me the right research foundation, and encouraging me to pursue a PhD in NLP. I would also like to thank my undergraduate friends, who have continued to be a great support system throughout my PhD, and often given me great outside perspectives on my work—in particular, Karan Chadha, Abhin Shah, Arka Sadhu, Siddhant Garg, Sudeep Salgia, Parth Kothari, Purva Tendulkar, Roshan Nayak and Sahil Bhatnagar.

Finally, I would like to thank to Yixiao, for being a wonderful friend and partner in the last two years. Our numerous mushroom and birding walks, movie nights, grocery shopping trips, and adventures in the Pioneer Valley and beyond are some of my most cherished PhD memories, and I am excited to continue our journey in the years to come. Thank you for listening to my practice job talk twenty times, and I promise I will never make you do that again! Last, but certainly not the least, I am grateful to my parents, sister and my grandmother for their unconditional love and support. Thank you for always inspiring me and letting me pursue my dreams thousands of miles away from home. I am excited to have finally joined you in the doctor’s club (at least somewhat)!
Text generation is an important emerging AI technology that has seen significant research advances in recent years. Due to its closeness to how humans communicate, mastering text generation technology can unlock several important applications such as intelligent chat-bots, creative writing assistance, or newer applications like task-agnostic few-shot learning. Most recently, the rapid scaling of large language models (LLMs) has resulted in systems like ChatGPT, capable of generating fluent, coherent and human-like text. However, despite their remarkable capabilities, LLMs still suffer from several limitations, particularly when generating long-form text. In particular, (1) long-form generated text is filled with factual inconsistencies to world knowledge and the input prompt; (2) it is difficult to accurately evaluate the quality of long-form generated text; (3) it is difficult to identify whether a piece of long-form text was AI-generated, a task necessary to prevent widespread misinformation and plagiarism.
In this thesis I design algorithms aimed at making progress towards these three issues in current LLMs. I will first describe a retrieval-augmented system we built for long-form question answering, to improve factual correctness of long-form generated text. However, a careful empirical analysis reveals issues related to input/output consistency of generated text, and an inherent difficulty in evaluation. I will then describe our model RankGen, which uses large-scale contrastive learning on documents to significantly outperform competing long-form text generation methods to generate text more faithful to the input. Next, I will describe our efforts to improve human evaluation of long-form generation (issue #2) by proposing the LongEval guidelines. LongEval is a set of three simple empirically-motivated ideas to make human evaluation of long-form generation more consistent, less expensive, and cognitively easier for evaluators. Finally, I describe my work on AI-generated text detection (issue #3), and showcase the brittleness of existing methods to paraphrasing attacks I designed. I will describe a simple new AI-generated text detection algorithm using information retrieval, which is significantly more robust to paraphrasing attacks.

Finally, I conclude this thesis with some future research directions that I am excited about, including plan-based long-form text generation, and a deeper dive into understanding large language model training dynamics.
TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>ACKNOWLEDGMENTS</th>
<th>v</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>xiv</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>xv</td>
</tr>
<tr>
<td>CHAPTER</td>
<td></td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Overview</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Roadmap</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Factual Consistency of Long-form Text Generation</td>
<td>3</td>
</tr>
<tr>
<td>1.4 Evaluation of Long-form Text Generation</td>
<td>4</td>
</tr>
<tr>
<td>1.5 Detecting AI-generated text</td>
<td>6</td>
</tr>
<tr>
<td>1.6 Future Directions</td>
<td>7</td>
</tr>
<tr>
<td>1.7 Declaration of Collaborations</td>
<td>8</td>
</tr>
<tr>
<td>2. LONG-FORM QUESTION ANSWERING: PROGRESS AND CHALLENGES</td>
<td>9</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>10</td>
</tr>
<tr>
<td>2.2 A state-of-the-art LFQA system</td>
<td>12</td>
</tr>
<tr>
<td>2.2.1 Retriever</td>
<td>12</td>
</tr>
<tr>
<td>2.2.2 Generator</td>
<td>13</td>
</tr>
<tr>
<td>2.2.3 Main Experiments</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Analysis</td>
<td>16</td>
</tr>
<tr>
<td>2.3.1 Are generations grounded in retrieval?</td>
<td>16</td>
</tr>
<tr>
<td>2.3.2 Training / Validation Overlap</td>
<td>20</td>
</tr>
<tr>
<td>2.3.3 ROUGE-L Bounds on ELI5 Performance</td>
<td>23</td>
</tr>
</tbody>
</table>
3. RANKGEN: IMPROVING TEXT GENERATION WITH LARGE RANKING MODELS ................................................. 29

3.1 Introduction ......................................................... 30
3.2 RANKGEN: a generation ranker ................................ 32
   3.2.1 LMs do not choose gold over negatives .......... 33
   3.2.2 Training RANKGEN ......................................... 35
   3.2.3 Using RANKGEN at inference ......................... 36

3.3 Experiments ......................................................... 37
   3.3.1 Model configurations ...................................... 38
   3.3.2 Open-ended text generation ............................... 39
   3.3.3 Results from automatic evaluations ................. 40
   3.3.4 Human evaluation with A/B tests .................... 42
   3.3.5 Text to Python code generation ...................... 42

3.4 Analysis ............................................................ 44
   3.4.1 In what ways does RANKGEN improve generation quality? .... 44
   3.4.2 How fast is decoding with RANKGEN? ............... 45
   3.4.3 Generation with different length prefixes ........... 46
   3.4.4 RANKGEN as a retriever ................................. 48

3.5 Related Work ....................................................... 48
3.6 Conclusion and Future Work .................................... 50

4. LONGEVAL: GUIDELINES FOR HUMAN EVALUATION OF FAITHFULNESS IN LONG-FORM SUMMARIZATION .......... 53

4.1 Introduction ....................................................... 54
4.2 Survey of human evaluation practices ........................ 56
4.3 The LONGEVAL guidelines for faithfulness human evaluation .......... 58
   4.3.1 RQ1: Does inter-annotator agreement improve using fine-grained annotations? .......................... 60
   4.3.2 RQ2: Can we reduce annotator workload by partially annotating a long summary? ......................... 65
   4.3.3 RQ3: Is it useful to align summary units to sentences in the source document? .............................. 66
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.3.4</td>
<td>To what extent do our findings generalize to short-form summarization?</td>
</tr>
<tr>
<td>4.4</td>
<td>Related Work</td>
</tr>
<tr>
<td>4.5</td>
<td>Conclusion</td>
</tr>
<tr>
<td>4.6</td>
<td>Limitations</td>
</tr>
<tr>
<td>4.7</td>
<td>Impact / Retrospective</td>
</tr>
<tr>
<td>5.1</td>
<td>Introduction</td>
</tr>
<tr>
<td>5.2</td>
<td>A background on detectors of AI-generated text</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Watermarking language model outputs</td>
</tr>
<tr>
<td>5.2.2</td>
<td>Statistical outlier detection methods</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Classifiers</td>
</tr>
<tr>
<td>5.2.4</td>
<td>Comparison to Sadasivan et al. (2023)</td>
</tr>
<tr>
<td>5.3</td>
<td>Building a controllable discourse paraphraser</td>
</tr>
<tr>
<td>5.4</td>
<td>Experiments attacking detection algorithms with DIPPER</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Evaluation metrics</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Models, datasets &amp; detection algorithms</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Attacking AI-generated text detectors</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Alternative paraphrasing attacks</td>
</tr>
<tr>
<td>5.5</td>
<td>Defense against paraphrase attacks using retrieval</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Formulating the retrieval defense</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Controlled comparisons of retrieval with other AI-generated text detectors</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Is retrieval an effective detector with a large retrieval corpus?</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Ideas to make retrieval detection work well at an even larger scale</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Limitations of retrieval for detection</td>
</tr>
<tr>
<td>5.6</td>
<td>Experiments measuring intrinsic paraphrase generation quality</td>
</tr>
<tr>
<td>5.6.1</td>
<td>Ablation studies on DIPPER</td>
</tr>
<tr>
<td>5.6.2</td>
<td>Human evaluation of semantic preservation using DIPPER</td>
</tr>
<tr>
<td>5.7</td>
<td>Conclusion</td>
</tr>
<tr>
<td>6.</td>
<td>FUTURE DIRECTIONS</td>
</tr>
</tbody>
</table>
6.1 Plan-based long-form generation .................................................. 106
6.2 Secure long-form generation ......................................................... 107
6.3 Understanding LLM training dynamics ......................................... 108
6.4 Evaluating recall in long-form text generation ............................. 109
6.5 Using fine-grained human evaluation in RLHF ................................. 110

BIBLIOGRAPHY .................................................................................. 111

APPENDICES

A. APPENDIX FOR “LONG-FORM QUESTION ANSWERING: PROGRESS
   AND CHALLENGES” ................................................................. 142
B. APPENDIX FOR “RANKGEN: IMPROVING TEXT GENERATION
   WITH LARGE RANKING MODELS” .............................................. 152
C. APPENDIX FOR “LONGEVAL: GUIDELINES FOR HUMAN
   EVALUATION OF FAITHFULNESS IN LONG-FORM
   SUMMARIZATION” ............................................................... 179
D. APPENDIX FOR “AI-GENERATED TEXT DETECTION:
   PARAPHRASING ATTACKS AND RETRIEVAL DEFENSES” ............ 188
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Results table on ELI5</td>
</tr>
<tr>
<td>2.2</td>
<td>Predicted retrieval vs random retrieval on our LFQA system</td>
</tr>
<tr>
<td>2.3</td>
<td>Human evaluation of LFQA system</td>
</tr>
<tr>
<td>2.4</td>
<td>Predicted retrieval vs random retrieval (fine-grained)</td>
</tr>
<tr>
<td>2.5</td>
<td>ELI5 train/validation overlap</td>
</tr>
<tr>
<td>2.6</td>
<td>KILT metrics upper and lower bounds</td>
</tr>
<tr>
<td>3.1</td>
<td>Motivation for InBOOK negatives</td>
</tr>
<tr>
<td>3.2</td>
<td>Motivation for GENERATIVE negatives</td>
</tr>
<tr>
<td>3.3</td>
<td>RankGen vs baselines using automatic evaluation</td>
</tr>
<tr>
<td>3.4</td>
<td>RankGen vs nucleus sampling, human evaluation</td>
</tr>
<tr>
<td>3.5</td>
<td>A comparison of RankGen with newer decoding methods (released in October 2022)</td>
</tr>
<tr>
<td>3.6</td>
<td>RankGen human evaluation inter-annotator agreement</td>
</tr>
<tr>
<td>3.7</td>
<td>RankGen performance on text to code generation</td>
</tr>
<tr>
<td>3.8</td>
<td>Distribution of human explanation types for RankGen evaluation</td>
</tr>
<tr>
<td>3.9</td>
<td>RankGen vs nucleus sampling - human explanation examples</td>
</tr>
<tr>
<td>3.10</td>
<td>RankGen retrieval performance</td>
</tr>
<tr>
<td>4.1</td>
<td>List of long-form summarization datasets in our LongEval survey</td>
</tr>
</tbody>
</table>
4.2 Human evaluation setups in prior summarization literature .................. 60
4.3 Inter-annotator agreement on fine-grained annotation vs coarse-grained. ..... 63
4.4 A comparison of hint highlighting algorithms tested in LongEval. ............. 67
4.5 Annotator performance in detecting summary errors with different types of highlight hints. ................................................................. 67
4.6 Annotator experience using highlighted hints in LongEval. .................... 68
5.1 Performance of detection algorithms after paraphrasing on open-ended generation. ................................................................. 85
5.2 Performance of detection algorithms after paraphrasing on long-form question answering. ......................................................... 87
5.3 A comparison of retrieval against other AI-generated text detectors on long-form QA. ................................................................. 92
5.4 DIPPER ablation studies compared to paraphrasing alternatives. ............. 99
5.5 Human evaluation of semantic preservation in DIPPER paraphrases. ....... 101
5.6 Outputs from DIPPER along with Likert ratings and annotator comments. ................................................................. 105
A.1 Effect of minibatch size on c-REALM ........................................... 144
A.2 Effect of weight initialization on c-REALM ...................................... 144
A.3 Number of parameters used by LFQA systems .................................. 145
A.4 ELIS5 performance on overlap vs non-overlap splits ......................... 146
A.5 Effect of generation truncation on ROUGE scores ............................. 147
A.6 Comparison of LFQA generations with predicted and random retrievals .... 148
A.7 Comparison of LFQA generations with predicted and random retrievals, \( p = 0.9 \) .............................................................................. 149
A.8 A fine-grained version of Table A.7 .................................................. 150
A.9 Human evaluation of LFQA — breakdown by question ...................... 150
A.10 Example generations from our LFQA system with $p = 0.9$. .............. 151
B.1 RankGen minibatch and model size ............................................. 152
B.2 Number of RANKGen calls in the same time as one LM generation ...... 154
B.3 A version of Table 3.1 with 10 distractors ................................. 157
B.4 RankGen retrieval performance on PG19 validation books ............. 158
B.5 Human vs generated text across domains/models ............................ 159
B.6 Zero-shot suffix identification with RankGen ............................. 160
B.7 RankGen evaluation using the rep metric .................................. 168
B.8 RankGen generation unigram overlap with prefix ....................... 168
B.9 Table B.8 but considering only lemmatized nouns .................... 169
B.10 Variation in suffix identification datasets with model / minibatch size ...... 169
B.11 Variation in PG19 / Wikipedia suffix identification with model / minibatch size .................................................................................................................. 170
B.12 RankGen ablation on MAUVE scores with different model / batch size ...... 170
B.13 Effect of RankGen minibatch / model size GENERATIVE negative identification .................................................................................................................. 171
B.14 Upwork job posting in RankGen ........................................ 172
B.15 More RankGen generations with preference explanations .............. 173
B.16 More RankGen model generations (part 1) ............................. 174
B.17 More RankGen model generations (part 2) ............................. 175
B.18 More RankGen model generations (part 3) ............................. 176
B.19 More RankGen model generations (part 4) ............................. 177
B.20 More RankGen model generations (part 5) ............................. 178
C.1 Inter-annotator agreement for FINE annotation per summary unit. ......... 180
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>C.2</td>
<td>Best practices in LONGEVAL surveyed papers.</td>
<td>182</td>
</tr>
<tr>
<td>C.3</td>
<td>Annotator backgrounds in LONGEVAL surveyed papers.</td>
<td>182</td>
</tr>
<tr>
<td>C.4</td>
<td>FINE-grained evaluation annotator guidelines in LONGEVAL.</td>
<td>186</td>
</tr>
<tr>
<td>C.5</td>
<td>COARSE-grained evaluation annotator guidelines in LONGEVAL.</td>
<td>187</td>
</tr>
<tr>
<td>D.1</td>
<td>Survey of previous paragraph-level paraphrasing papers</td>
<td>190</td>
</tr>
<tr>
<td>D.2</td>
<td>Performance of retrieval-based text detection on Wikipedia prompts.</td>
<td>191</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>Example of factual issues in generated text</td>
<td>3</td>
</tr>
<tr>
<td>1.2</td>
<td>Issues with evaluating long-form generation</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>Summary of hurdles in long-form QA</td>
<td>11</td>
</tr>
<tr>
<td>2.2</td>
<td>Generation example by our LFQA system</td>
<td>16</td>
</tr>
<tr>
<td>3.1</td>
<td>RankGen training time setup</td>
<td>31</td>
</tr>
<tr>
<td>3.2</td>
<td>RankGen setup during inference</td>
<td>37</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance / time tradeoff across RankGen hyperparameters</td>
<td>47</td>
</tr>
<tr>
<td>3.4</td>
<td>RankGen performance with shorter length prefixes</td>
<td>47</td>
</tr>
<tr>
<td>4.1</td>
<td>Overview of research questions considered in LONGLONGEVAL</td>
<td>55</td>
</tr>
<tr>
<td>4.2</td>
<td>FINE vs COARSE, Pearson correlation between automatic / human evaluation</td>
<td>61</td>
</tr>
<tr>
<td>4.3</td>
<td>FINE vs COARSE, average model performance</td>
<td>61</td>
</tr>
<tr>
<td>4.4</td>
<td>Empirical benefits of partially annotating a long summary.</td>
<td>64</td>
</tr>
<tr>
<td>5.1</td>
<td>An overview of DIPPER paraphrasing attacks on watermarked text</td>
<td>76</td>
</tr>
<tr>
<td>5.2</td>
<td>The algorithm used to train DIPPER</td>
<td>82</td>
</tr>
<tr>
<td>5.3</td>
<td>An illustration of AI-generated text detection with retrieval</td>
<td>91</td>
</tr>
<tr>
<td>5.4</td>
<td>Retrieval detection rate variation with corpus size and query length.</td>
<td>93</td>
</tr>
<tr>
<td>5.5</td>
<td>ROC plots for GPT2-XL using different detectors, before and after paraphrasing</td>
<td>94</td>
</tr>
</tbody>
</table>
5.6 The interface of the annotation platform used in our human study. ........... 102

A.1 Attention maps in routing transformer .............................................. 143

A.2 Scatter plot between ROUGE-L scores and unigram overlap with all retrievals ................................................................. 148

A.3 Scatter plot between ROUGE-L scores and unigram overlap with retrievals which were correct ................................................. 149

B.1 Simplified RankGen decoding implementation in Python ................. 164

B.2 RankGen performance tradeoff across hyperparameters .................. 165

B.3 RankGen timing analysis across hyperparameters ......................... 165

B.4 MAUVE divergence curves using RankGen-XL-all .......................... 166

B.5 MAUVE divergence curves using RankGen-XL-PG19 ....................... 167

B.6 RankGen human evaluation interface .............................................. 171

C.1 Learning effect over time while evaluating long-form summaries with FINE annotation .............................................................. 183

C.2 A version of Figure 4.4 using Pearson correlation instead of Kendall Tau correlation ................................................................. 184

C.3 A version of Figure 4.2 using Kendall’s Tau correlation instead of Pearson’s correlation .............................................................. 184

C.4 The AMT Sandbox annotation interface used for FINE evaluation of SQuALITY and PubMed summaries (Appendix C.2.1) .... 185

C.5 The LabelStudio annotation interface used for COARSE evaluation of PubMed summaries (Appendix C.2.2) .......................... 185

D.1 More ROC plots for GPT2-XL using different detectors, before and after paraphrasing .............................................................. 192
CHAPTER 1
INTRODUCTION

1.1 Overview

Natural language generation (NLG) is broadly defined as an algorithm which outputs natural language (a sentence, paragraph or document) given an input requirement. Several software applications fall under the broad umbrella of NLG, including machine translation, text summarization, story generation, dialogue agents and long-form question answering. NLG technology has seen significant advances in the last 2-3 years, most notably due to the introduction of large language models (LLMs) like GPT-2,3,4 [208, 23, 225, 195, 24] which are trained to predict the next word on internet-scale data. Modern LLMs are capable of generating fluent, coherent, and human-like text, quite unlike any AI-system preceding it. However, despite their remarkable capabilities, LLMs still suffer from several limitations, particularly when generating long-form text (defined below). In this thesis I focus on identifying and making progress towards three such limitations: (1) factual inconsistencies in long-form generated text; (2) inherent difficulty in evaluating long-form generated text; (3) difficulty in identifying whether text was AI-generated.

What is long-form text generation? In this thesis, we define a long-form text generation algorithm as, “Any algorithm that outputs natural language sequences which are between 100 and 300 words in length.” Intuitively, this corresponds to the process of generating 1-3 paragraphs of text and up to the length of a short document, similar to typical outputs from modern chatbots like ChatGPT [225] or Bard [202]. Several important natural language processing tasks fall within this broad definition, including but not limited to, long-form
question answering, scientific abstract generation, story generation, text summarization. An important feature of long-form text generation is its *abstractive* nature: the desired output sequences are not already present in human-written forms on the internet. Algorithms must synthesize novel outputs from scratch, using their background knowledge or information present in the input prompt.

The techniques developed in this thesis are not necessarily limited to this definition, and they may generalize to longer sequences or other modalities (for example, we also evaluate Python source code generation in Chapter 3). However, as we will discuss in Chapter 4, evaluation becomes extremely challenging for long output sequences, and we restrict sequences to the 100-300 word range for more manageable evaluation.

### 1.2 Roadmap

In this thesis I focus on identifying some key issues in current long-form text generation methods, and develop algorithmic solutions aimed at addressing them. I focus on three issues in current long-form text generation algorithms, which are powered by large language models. These issues are listed below, along with the corresponding chapters:

- **Issue #1**: Long-form generated text is filled with factual inconsistencies related to world knowledge and the input prompt (Chapter 2, Chapter 3);

- **Issue #2**: It is difficult to accurately evaluate the quality of long-form generated text, especially aspects like factual correctness; (Chapter 2, Chapter 4);

- **Issue #3**: It is difficult to identify whether a piece of long-form text was AI-generated, a task necessary to prevent widespread misinformation and plagiarism (Chapter 5).

Every chapter listed above begins with a short motivation / chapter overview connecting it to the rest of the thesis, before the actual technical content. Additionally, each chapter concludes with an “Impact / Retrospective” section, which is a discussion of its downstream
Figure 1.1: An example of factual issues in text generated by GPT-3 [23] in the task of long-form question answering. Current models suffer from two kinds of issues which are collectively termed as “hallucinations” — (1) factually inconsistent to global facts; (2) factually inconsistent or irrelevant to the input.

1.3 Factual Consistency of Long-form Text Generation

A major limitation of current text generation algorithms is their tendency to “hallucinate” (Figure 1.1). Current models often produce text which is either factually incorrect or unfaithful to the input. This problem has been noticed across different tasks, including data-to-text generation [291], text summarization [175], style transfer [134], and open-ended text generation [46]. This is a persistent problem [199, 182, 62] even in the latest large
language models like ChatGPT, PaLM and GPT-4, as well as language models utilizing search engines [163], such as Bing Chat or Perplexity AI.¹

In this thesis I develop methods aimed at identifying and tackling this issues, focusing on long-form question answering and open-ended text generation. In Chapter 2, I study the problem of long-form question answering (LFQA), where a system needs to generate a paragraph-long answer to Why or a How question. In order to improve the factual correctness of the generated text, I develop an algorithm that uses retrieval to ground model generations in Wikipedia articles. Our proposed method achieves a new state-of-the-art on the ELI5 benchmark [57]. However, a careful analysis reveals several troubling issues with our model, the ELI5 dataset and LFQA evaluation. Most notably, I notice that the model’s generations are often unrelated (“unfaithful”) to the retrieved documents which are supplied as inputs.

Can we make long-form generated text more faithful to the input? In Chapter 3 I introduce RankGen, an algorithm designed to improve the consistency between the input and generated text. RankGen is a 1.2B parameter encoder network which maps inputs and generations to a shared vector space. The dot product similarity between the input and the generation vectors scores the generation. I evaluate RankGen on open-ended generation and Python source code generation, and find that it significantly improves over competing algorithms like nucleus sampling [89] and typical sampling [176].

1.4 Evaluation of Long-form Text Generation

Robust evaluation protocols are critical to build better algorithms. Unfortunately, text generation is hard to evaluate due to the large number of possible valid outputs for the same input (example in Figure 1.2). Evaluation is particularly hard for generations longer than a sentence, since metrics need to be able to reward partially correct inputs.

¹https://perplexity.ai
Figure 1.2: The issues with evaluating long-form generation. The red boxes show a small subset of possible ground truth answers. Due to the large output space in a task like long-form question answering, evaluation is challenging.

In Chapter 2 I analyze the evaluation protocols for long-form question answering, where ROUGE [161] is used to rank systems on the official leaderboard. I find that simple heuristics such as input copying got higher ROUGE scores than several systems (like RAG [155]) which actually perform the task. While human evaluation is generally regarded a gold standard in natural language generation [26], even human evaluation of long-form generated text is considered difficult and time consuming to evaluate. Annotators are often not incentivized to perform evaluation carefully, and often disagree with each other’s judgments [112].

In Chapter 4, I first survey the evaluation practices in 162 papers studying long-form text generation and find a grim situation in prior work: a) 73% paper did not perform human evaluation and solely rely on flawed automatic metrics; b) the other 27% severely lacked in human evaluation best practices [64], often not reporting crucial information (like annotator instructions) needed for reproducibility. This issue is even more relevant in the LLM-era, since human evaluation is being extensively used to improve LLMs with reinforcement learning [196, 225, 258].
To move towards more robust human evaluation, in Chapter 4 I propose the LongEval guidelines, three simple empirically justified techniques to improve human evaluation of factual correctness in long-form text summarization. Through carefully conducted annotator studies on long-form generated text, I show the benefit of 1) evaluating long-form generations at a clause level for high inter-annotator agreement; 2) randomly sampling a fraction of clauses in the generation to save annotation cost; 3) automatically aligning clauses with sentences in the source document to help annotators assess the clause’s factual correctness.

### 1.5 Detecting AI-generated text

Chapter 2 to 4 focus on improving the correctness of long-form generated outputs, and developing better methods to evaluate it. These topics are currently receiving extensive research interest, and the technology is rapidly improving. Given the rapid progress, there is a growing possibility that AI-generated long-form text will soon be indistinguishable from human-written text [24, 223]. While this may greatly benefit society, there is a growing fear of malicious use cases, such as fake content creation and academic plagiarism. This has motivated the development of approaches that identify AI-generated text, including those based on watermarking [121] or outlier detection [184].

In Chapter 5, I first show that all these AI-generated detection algorithms can be evaded by paraphrasing attacks, where the input is rewritten to preserve input semantics but using different lexical and syntactic properties [17, 134]. To perform this attack, I design a specialized paraphraser (DIPPER), which can paraphrase long-form inputs and provide fine-grained diversity control to paraphrase text just enough to just evade the detector. Paraphrasing with DIPPER drops the detection rate of a popular detection algorithm DetectGPT [184] from 70.3% to 4.6%, without appreciably modifying semantics.

Furthermore in Chapter 5, as a effective defense against DIPPER, I propose a new AI-generated text detection algorithm which uses information retrieval. Given a candidate text, this algorithm searches the database of sequences previously generated by the API,
searching for sequences matching the candidate text within a certain threshold. I show that retrieval is robust to paraphrasing attacks, and can detect 80-97% paraphrases across different settings, even when the database has over 15M entries.

1.6 Future Directions

Finally, in Chapter 6, I will conclude this thesis by outlining several directions for future research in long-form text generation. Some of these ideas are directly motivated by chapters of this thesis, while others are directions I personally find exciting. These directions include: (1) Planning in long-form text generation; (2) evaluating recall in long-form text generation; (3) using fine-grained evaluation feedback to improve language models with RLHF; (4) secure long-form text generation; (5) better understanding language model training dynamics.

1.7 Declaration of Collaborations

The following research papers are described in this thesis. They are produced in collaboration with the researchers listed below and have been published or submitted for publication:


• **Kalpesh Krishna**, Yixiao Song, Marzena Karpinska, John Wieting, Mohit Iyyer.

“Paraphrasing evades detection of AI-generated text, but retrieval is an effective defense” in arXiv and in submission to NeurIPS 2023 [132].
CHAPTER 2
LONG-FORM QUESTION ANSWERING: PROGRESS AND CHALLENGES

Motivation and Summary: As discussed in Chapter 1, large language models are becoming adept at generating fluent and coherent human-like text. However, their outputs are often riddled with inconsistencies with respect to world knowledge and to the input prompt (“hallucination” errors). Moreover, these errors are difficult to detect with automatic metrics, due to the long-form nature of the generated text. This chapter will empirically describe these issues in long-form question answering, and lay the foundation for future chapters on reducing hallucination (Chapter 3), and improving evaluation (Chapter 4) of long-form generated text.

This chapter studies long-form question answering (LFQA), which is the task of answering an information seeking Why or a How question (like How do jellyfish function without brains or a nervous system?) using a paragraph long answer. I will first design a new retrieval-augmented system that uses contrastive learning and sparse attention to find relevant information from Wikipedia which can answer a question, thereby attempting to reduce factual errors in generated text. While the developed system tops the public leaderboard, a detailed analysis will reveal troubling trends: (1) our system’s generated answers are not actually grounded in the documents that it retrieves; (2) ROUGE-L is not an informative metric of generated answer quality and can be easily gamed; and (3) human evaluations are difficult due to the long length and technical nature of generated text. ¹

¹Code accompanying this chapter can be found in https://github.com/martiansideofthemoon/hurdles-longform-qa.
2.1 Introduction

Long-form question answering (LFQA) integrates the retrieval component of open-domain QA, which involves searching a large external knowledge source for documents relevant to a given question, with a text generation component to produce paragraph-length answers. Significant progress has been made on open-domain QA datasets such as Natural Questions [142], whose questions are answerable with short phrases and entities, by leveraging dense retrieval techniques like ORQA [151], REALM [80], and DPR [114, 155, 102]. Methods inspired by these results have recently been combined with pretrained language models [154, 200] and applied to the Reddit-derived “Explain Like I’m Five” (ELI5) dataset [57], the largest publicly-available large-scale LFQA dataset.

The recently proposed KILT benchmark [200], which compares retrieval-augmented models across a variety of knowledge-intensive tasks including ELI5, automatically evaluates LFQA models by the quality of both generated answers (ROUGE-L against reference answers) and retrieved documents (R-precision against human-annotated relevant documents). In this chapter, we build a state-of-the-art system\(^2\) for ELI5 by using a sparse Transformer variant [222] to condition over Wikipedia paragraphs returned by a REALM-style retriever [80].

However, despite its success on the KILT leaderboard, our system does not actually use the documents that it retrieves! To measure the effect of retrieval on generation quality, we design a control experiment in which retrieved documents are replaced with randomly-sampled documents at inference time. Results from both human A/B tests and automatic metrics like ROUGE-L demonstrate that conditioning on random documents has almost no effect on generated answer quality (Figure 2.1c). We recommend that future LFQA research report the results of such control experiments in addition to reporting generation and retrieval quality.

\(^2\)At the time of submission (23rd November, 2020), https://evalai.cloudcv.org/web/challenges/challenge-page/689/leaderboard/1908
Val Q: Can you protect electronics from EMPs/solar flares? If so, how?

Generation using random retrievals, 24.8 ROUGE-L

Yes, you absolutely can, in fact you can build a Faraday cage around your electronics, and protect them from solar flares... This is what is done with the Faraday cage around your electronics, which is the problem. The reason it is expensive is because it requires a huge amount of power and is expensive to replace... designed to shield your electronics from solar flares, you will have to pay for the protection. This is because you have to buy a piece of equipment that is designed to be as cheap as possible...

Random Train Ans, 19.4 ROUGE-L

The fast lane/slow lane is a bit of a misnomer. It gives the impression that new, faster lanes are being built. In reality, normal speed will be...

Gold Answer, 18.6 ROUGE-L

I'll start with the grounding question, because that's the easiest to answer: Doesn't help a bit. All that matters is that the metal container is conductive and doesn't have gaps... completely seal your Faraday cage. Consider soldering the lid on to that paint can... look at little baggie it comes in. Sealed mylar. That protected that chip from air travel at 35,000 feet, land travel through rural, urban, and suburban areas, and all the electromagnetic radiation that the trip entails... No lead shielding. No safes....

Generation using predicted retrievals, 19.0 ROUGE-L

Yes, you can shield them. But it's a slow process... Also, the equipment that's powered by them is a lot more expensive than you'd think, so it's hard to make sure that you're not just shielding them from your remote control.

Generation using random retrievals, 24.8 ROUGE-L

Yes, you absolutely can, in fact you can build a Faraday cage around your electronics, and protect them from solar flares... This is what is done with the Faraday cage around your electronics, which is the problem. The reason it is expensive is because it requires a huge amount of power and is expensive to replace... designed to shield your electronics from solar flares, you will have to pay for the protection. This is because you have to buy a piece of equipment that is designed to be as cheap as possible...

Figure 2.1: A summary of the major hurdles (a-d) to progress in long-form question answering with ELI5.

How can a system using random retrieval perform well on ELI5? Our analysis reveals that this result is partially due to significant train / validation overlap in the ELI5 dataset (Figure 2.1a), which eliminates the need for external retrieval. A human study shows that at least 81% of validation questions have a paraphrase in the training set, and almost all validation questions are topically similar to a training set question. While [57] attempted to identify and remove question overlap using TF-IDF similarity, more complex semantic matching methods & human verification is needed to address this issue in future LFQA datasets.

Digging deeper, we identify fundamental issues with using ROUGE-L to evaluate generated answer quality (Figure 2.1b). Simple baselines such as just repeatedly copying the question, or choosing a random training set answer, can outperform LFQA systems such as RAG [155] in terms of ROUGE-L. On the other hand, our system achieves higher ROUGE-L than reference human-written answers, which is misleading since human A/B testers strongly prefer reference answers to our system’s. We conclude that ROUGE-L is
not a reliable metric to evaluate LFQA due to its large and relatively unconstrained output space (e.g., compared to translation or summarization), and we offer suggestions for better automatic & human evaluations to enable meaningful progress on this task.

2.2 A state-of-the-art LFQA system

The ELI5 task [57] asks models to generate paragraph-length answers to open-ended questions that often rely on world knowledge (e.g., how do jellyfish function without brains or nervous systems?). LFQA systems thus benefit from conditioning answer generation on relevant documents from the web (such as the Wikipedia article about jellyfish). While large-scale pretrained language models store surprising amounts of world knowledge within their parameters [201, 219], external document retrieval not only augments this intrinsic knowledge but also grounds model outputs in a knowledge source, which provides interpretability.

In this section, we describe our proposed LFQA system, which conditions answer generation on Wikipedia articles identified by a pretrained retriever. We use a dense retriever trained by scaling up a distantly supervised algorithm from [106]. Since retrieved articles can be quite long and often exceed the maximum sequence length of pretrained models like BERT [42], we use a sparse-attention variant of the Transformer to allow modeling over longer sequences. Our system sets a new state of the art on ELI5, although we question the significance of this result in Section 2.3.

2.2.1 Retriever

We begin by specifying our dense retriever (“contrastive REALM” or C-REALM), which returns documents related to an input question. Consider a corpus of long-form questions and answers, represented by \((q_i, a_i)_{i=1}^N\). Our retriever uses \(q_i\) as a query to retrieve \(K\) documents \((r_{i,j})_{j=1}^K\) from a knowledge corpus (Wikipedia), which is enabled by an encoder network that projects both questions and candidate documents to a 128-\(d\) shared
embedding space. Like REALM [80], our encoder is a BERT-base Transformer [42] with a final projection layer.

Since the ELI5 dataset does not include gold retrievals, we train our retriever by scaling up a method recently introduced by [106] that uses gold answers for distant supervision. The key idea is to push the encoded vector for a question close to a vector representation of its ground-truth answer(s), but away from all other answer vectors in the mini-batch (negative examples). Intuitively, this method works because both ELI5 answers and external documents are of paragraph length (documents are paragraph-length chunks from Wikipedia). Concretely, we optimize the loss,

\[
\text{loss} = - \sum_{(q_i, a_i) \in B} \log \frac{\exp q_i \cdot a_i}{\sum_{a_j \in B} \exp q_i \cdot a_j}
\]

where \( B \) is the mini-batch and \( q_i, a_i \) are the encoded vector representations for \((q_i, a_i)\).

This objective is based on contrastive learning, a method that has been used effectively for semi-supervised learning [30] and dense retriever training [114]. Scaling up from [106], we use large mini-batches of size 12K, and initialize our retriever with the REALM CCNews checkpoint [80].³ These design decisions greatly improve retriever quality, as we observe in an ablation study (see Appendix A.3). During inference, a MIPS search is conducted with the ScaNN library [77] to efficiently find the top \( K \) documents; we use \( K = 7 \) in all experiments.

### 2.2.2 Generator

We next describe our generator model, which conditions its generated answers on retrieved documents returned by c-REALM. We use the Routing Transformer (RT) from [222], which is the current state of the art in long-form language modeling. The RT is a sparse attention model that employs local attention as well as mini-batch \( k \)-means clustering to better...

³[106] use batch size 512, initialize with BERT.
model long-range dependencies in sequences (attention maps in Appendix A.2). Long-form
language models such as RT are well-suited to ELI5 as the task requires conditioning answer
generation not only on a short question but also many lengthy retrieved documents.

We pretrain our RT model on PG-19, a long-form language modeling benchmark [210]
created from approximately 28,000 Project Gutenberg books published before 1919. PG-19
has 1.9B tokens and an average context size of 69K words. While this data is out-of-domain
for ELI5, we choose it to encourage long & coherent generation. Our RT is a 22-layer model
with 1032 hidden units (486M parameters), maximum sequence length of 8192 tokens, and a
vocabulary of 98K subwords.\(^4\) We fine-tune our model in a decoder-only fashion [164, 294]
by concatenating the top \(K\) retrieved documents to the question \([r_{i,K}, r_{i,K-1} \ldots r_{i,1}, q_i, a_i]\)
and training the model to predict tokens of the answer \(a_i\). We do not backpropagate gradients
through the retriever.\(^5\) Retrievals slightly improve perplexity (18.1 vs 17.8) as seen in [276],
but do not improve generations (§2.3.1).

2.2.3 Main Experiments

Dataset & Evaluation: We evaluate our model on the KILT validation & test subsets of
ELI5 [200], since the original ELI5 dataset does not have human annotations to measure
retriever performance (split sizes in Appendix A.1). Answer quality is measured by the
maximum overlap of generations with a set of gold answers in terms of unigram F1 score and
ROUGE-L [161]. [200] collected human annotations of Wikipedia articles which support
ELI5 gold answers, which enables measuring retrieval quality by computing R-precision (if
the top-1 retrieval matches the annotation) and Recall@5 using the top-5 retrievals. Finally,
the KILT benchmark combines R-prec. and ROUGE-L to measure the overall performance
of the system by “KILT ROUGE-L”. This metric is similar to ROUGE-L, but assigns a score

\(^4\)Our hyperparameters have been chosen manually with minimal tuning. See Appendix A.2 for details.
\(^5\)We tried training the retriever jointly with RT using the attention bias scheme proposed in MARGE [153].
This improved perplexity only in autoencoding settings where the gold answer itself is used as a retrieval
query (like the setup in [153]), which is not valid in LFQA.
Table 2.1: Results on the KILT test set for ELI5 for (1) retrieval performance, using R-precision and Recall@5 (RPrec, R@5), and (2) generation quality, using ROUGE-L (R-L). These scores are combined to produce the final metric KILT R-L (KRL). We outperform prior work on both generation & combined scores.

<table>
<thead>
<tr>
<th>Model</th>
<th>Retrieval</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RPr.</td>
<td>R@5</td>
</tr>
<tr>
<td>T5-base</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>BART</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>RAG</td>
<td>11.0</td>
<td>22.9</td>
</tr>
<tr>
<td>BART + DPR</td>
<td>10.7</td>
<td>26.9</td>
</tr>
</tbody>
</table>

\[ p = 0.9 \]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RT + REALM</td>
<td>6.7</td>
<td>15.5</td>
<td>25.1</td>
<td>21.5</td>
<td>1.4</td>
</tr>
<tr>
<td>RT + C-REALM</td>
<td>10.2</td>
<td>24.4</td>
<td>25.4</td>
<td>21.5</td>
<td>2.1</td>
</tr>
</tbody>
</table>

\[ p = 0.6 \]

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>RT + REALM</td>
<td>6.7</td>
<td>15.7</td>
<td>23.1</td>
<td>23.4</td>
<td>1.5</td>
</tr>
<tr>
<td>RT + C-REALM</td>
<td>10.7</td>
<td>24.6</td>
<td>22.9</td>
<td>23.2</td>
<td>2.4</td>
</tr>
</tbody>
</table>

of 0 whenever the top-1 retrieval does not match the gold annotation.

**Baselines:** We compare our model with the other entries on the ELI5 KILT leaderboard which are either generation-only, like T5-base [211] and BART [154], or variants of BART using retrieval such as RAG [155] and BART + DPR [200]. These systems are based on massive pretrained language models, with similar number of parameters as our model (details in Appendix A.4).

**Results:** Table 2.1 contains our results on the test set of the ELI5 (also on the public KILT leaderboard). We present four variants of our system, using a different retriever during inference (REALM or c-REALM), and different nucleus sampling \( p \) values [89]. All variants outperform prior work in generation quality, with lower-entropy models \( (p = 0.6) \)
Q: Why are almost all boats white?

A: Boats are generally painted white for obvious reasons, as they are inexpensive and easy to clean. Ships are also often painted to provide a visible visual contrast to the sea. Boats also often have high visibility ports to allow for easy international trade. In addition, many boats have an air-breathing function that makes them easier to spot and stay afloat, which is a good thing if you’re a pirate or trying to get into some country’s waters. Lastly, many ships are built in the last few decades, so they are not as exotic looking as they were thirty years ago. And lastly, many ships are made out of materials that are easy to work with, and so many of them are painted white as a practical matter. There are many, many more reasons for why ships are white, but those are the ones I know of. I’m sure there are many other reasons, but those are the ones I know of. Edit: I forgot to mention that the other answers about visibility and easy to see ports are also true. Ships are painted to appear to be larger, and more expensive. That makes them appear more impressive, and so they are more likely to be bought by people who want to show off their wealth and status.

Figure 2.2: Example generation from our LFQA system with \( p = 0.9 \). Generations are long & coherent, but suffer from repetition towards the end. (more in Appendix A.5 and attached data supplementary material).

performing best.\(^6\) c-REALM performs competitively to RAG and DPR despite being only distantly supervised, and outperforms REALM. Our proposed RT+c-REALM system achieves a new state-of-the-art on combined performance (KILT R-L). Generations from our model are provided in Figure 2.2 and Appendix A.5.

2.3 Analysis

In this section, we conduct a thorough analysis of our model’s usage of retrievals (Section 2.3.1), the impact of overlap in ELI5’s train / validation / test folds (Section 2.3.2), issues with ROUGE-L and performance bounds (Section 2.3.3), and the difficulty in human evaluation for this task (Section 2.3.4).

2.3.1 Are generations grounded in retrieval?

While our retrieval-augmented system achieves state-of-the-art performance, we find little evidence that it is actually using the retrieved documents. To measure this, we run an ablation study where at inference time we replace retrieved paragraphs with randomly

\(^6\)As in [89], a human study reveals that higher entropy \( (p = 0.9) \) answers are slightly more coherent and sensible, but lower entropy answers \( (p = 0.6) \) are more relevant to the question (details in Appendix A.7).
Table 2.2: Comparison of generations (with $p = 0.6$) conditioned on predicted retrievals (Predicted) and randomly chosen retrievals (Random). Notice small differences in: (1) ROUGE-L vs gold answers (R-L); (2) $n$-gram overlap ($n$-g) with predicted retrievals (vs predicted retr.). Gold answers also have a similar overlap with predicted retrievals. To control for stopwords, we show overlaps with the random retrievals.

<table>
<thead>
<tr>
<th></th>
<th>vs predicted retr.</th>
<th>vs random retr.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-L</td>
<td>1-g</td>
</tr>
<tr>
<td>Predicted</td>
<td>24.42</td>
<td>52.3</td>
</tr>
<tr>
<td>Random</td>
<td>24.20</td>
<td>51.2</td>
</tr>
<tr>
<td>Gold Ans</td>
<td>-</td>
<td>54.1</td>
</tr>
</tbody>
</table>

Table 2.3: Human evaluation results with exact number of ratings shown in (·). Annotators are shown a question along with two answers (A, B) in random order and ask them to choose one (details in Appendix A.7). For both model variants ($p = 0.6, 0.9$), we see (1) little difference between generations conditioned on predicted (pred.) or random (rand.) retrievals; (2) strong preference for gold answers over generations.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Prefer A</th>
<th>Prefer B</th>
<th>Tie</th>
</tr>
</thead>
<tbody>
<tr>
<td>For $p = 0.6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pred. random</td>
<td></td>
<td>40% (78)</td>
<td>33% (64)</td>
<td>27% (51)</td>
</tr>
<tr>
<td>pred. gold ans.</td>
<td></td>
<td>14% (29)</td>
<td>68% (138)</td>
<td>18% (36)</td>
</tr>
<tr>
<td>For $p = 0.9$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pred. random</td>
<td></td>
<td>31% (52)</td>
<td>37% (63)</td>
<td>32% (54)</td>
</tr>
<tr>
<td>pred. gold ans.</td>
<td></td>
<td>17% (49)</td>
<td>72% (203)</td>
<td>11% (31)</td>
</tr>
</tbody>
</table>

sampled paragraphs from Wikipedia. We compare this Random baseline with our original system (Predicted) in terms of generation quality as well as the $n$-gram overlap between the generation and the retrieved paragraphs.

**Generations are similar irrespective of type of retrievals:** We present our results in Table 2.2. Despite not being conditioned on any meaningful retrievals, the Random retrieval model has similar ROUGE-L scores as our Predicted system. Moreover, generations from the Random and Predicted models have similar amounts of 1-gram and 2-gram overlap with
Table 2.4: A fine-grained version of Table 2.2 measuring the unigram overlap of nouns/numbers in the generations with the input question (vs qn.), retrievals predicted by c-REALM (vs predicted retr.) and randomly sampled retrievals (vs random retr.). Similar to Table 2.2, notice very little difference with and without retrieval.

The $n$-gram overlaps are possibly overestimates due to stopwords (e.g., prepositions, punctuation) and entities which are copied from the question. To tackle this issue, in Table 2.4 we measure the fractions of lemmatized nouns, proper nouns and numbers in the generated answer which are present in the predicted retrievals but not in the question. We notice similar trends as before, with only small differences between the two systems. Finally, there is almost no correlation (Spearman $\rho = 0.09$) between the Predicted model’s generation quality and the amount of unigram overlap between its outputs and the retrieved documents (scatter plots in Appendix A.9), strengthening our hypothesis that generations are not grounded in retrievals.

**Human evaluation validates our findings:** As ROUGE-L and $n$-gram overlap have major limitations for LFQA (Section 2.3.3), we perform additional human A/B testing on the

---

7 Corresponding experiments with the $p = 0.9$ variant of our model are presented in Appendix A.9.

8 All these trends persist even on questions for which our retriever predicts the ground-truth document (Appendix A.9)
output of Random and Predicted. Specifically, we ask human volunteers\(^9\) to choose between answers generated by the two systems (presented in random order). As seen in Table 2.3, humans struggle to choose which of the two answers is more relevant to the question. For both model variants \((p = 0.6, 0.9)\), there is a less than 7% preference for a particular answer type, with humans preferring answers (by 6%) from the Random model for \(p = 0.9\)!

**Other systems also have this issue, possibly due to source-reference divergence and train-validation overlap:** We note that this issue is not unique to our system — other systems on the KILT leaderboard like BART + DPR and RAG actually perform worse than their no-retrieval counterpart (BART) in generation quality, as shown in Table 2.1. Qualitatively, we found no evidence of retrieval usage in a publicly hosted ELI5 model demo by [106].\(^{10}\) A possible explanation for this issue is high source-reference divergence, a common problem in table-to-text generation [291, 253]. In Table 2.2 and Table 2.4, we measure the \(n\)-gram overlap of top-ranked gold validation answers (Gold Ans) with predicted retrievals. This overlap is low and similar to that of our generations, which we suspect encourages our model to ignore retrievals. A second explanation is the large amount of train-validation overlap (Section 2.3.2), which eliminates the need for retrieval.

**Why does our model do well compared to other systems despite not using retrievals?** While our model has similar capacity as the BART/RAG baselines (comparison in Appendix A.4), we hypothesize that our improvements in ROUGE-L are due to a different pretraining objective. BART is pretrained on a masked infilling task on short sequences. Instead, we pretrain our model to perform next-word prediction on long sequences from Project Gutenberg, which encourages long & fluent generations. To illustrate this length effect, in Appendix A.8 we show that truncated outputs from our model get lower ROUGE-L

\(^9\)Details of our experimental setup in Appendix A.7.

\(^{10}\)https://huggingface.co/qa
scores on ELI5.\footnote{While we do not have access to generations from baselines on the KILT leaderboard, example generations from the demo of the BART model in [106] are significantly shorter (59 words avg.) than our generations (187 words avg.).} Prior summarization literature [243] has also shown that ROUGE scores vary heavily by length. To compare the same systems on shorter length outputs, we also tried finetuning the pretrained model on Wizard of Wikipedia [44], an unconstrained dialogue generation task with \textbf{single sentence} dialogues (much shorter than ELI5). As seen on the public KILT leaderboard,\footnote{https://eval.ai/web/challenges/challenge-page/689/leaderboard/1909} our system has \textit{lower} ROUGE-L scores than the BART / RAG baselines. Another possible explanation is issues with ROUGE-L itself, as discussed in Section 2.3.3.

\textbf{Takeaway (better evaluation of grounding):} For evaluating LFQA, it is important to run control experiments with random retrievals \& measure grounding of generations in retrieval. While the KILT benchmark does attempt to measure the combined retrieval + generation performance via KILT RL, it does not check whether the generations actually \textit{used} the retrievals. In other words, one can submit independent retrieval \& generation systems, but still perform well on the combined score. This may not be an issue for short-form QA tasks like Natural Questions, since the gold answer is often exactly contained as a span in the gold retrieval. Also, as retrieval might be less important for large language models with parametric knowledge [219], the KILT-RL strategy of simply aggregating top-1 retrieval score with ROUGE-L unfairly penalizes systems not relying on retrieval.\footnote{Another issue of KILT-RL is ignoring non top-1 retrievals, penalizing models using multiple retrievals together in context.}

\textbf{2.3.2 Training / Validation Overlap}

Our experiments in Section 2.3.1 show that model performance is mostly unchanged by conditioning generation on randomly sampled retrievals instead of predictions from C-
REALM. Despite not using retrievals, we observe qualitatively that our model displays a large amount of parametric knowledge ("Faraday Cage" in Figure 2.1c), which is surprising since it was pretrained on novels from Project Gutenberg (not Wikipedia). In this section, we discover that a major reason for ignoring retrievals is the large amount of train / validation overlap in ELI5. While [57] attempted to fix this issue through TF-IDF overlap, this method is insufficient to identify all question paraphrases, as we find significant overlap between the training set and the KILT validation set of ELI5.\textsuperscript{14} ELI5 is not the only dataset with substantial train / test overlap: [156] identify similar issues with short-form QA datasets like Natural Questions.

**Finding similar questions & measuring overlap:** We use our retriever c-REALM to retrieve similar questions from the training set, since it has learned to map questions to a feature-rich embedding space. For each validation question, we retrieve the 7 most similar training set questions. We use both human and automatic evaluation to calculate the amount of overlap. For human evaluation, we show annotators on Amazon Mechanical Turk\textsuperscript{15} a validation set question and a retrieved training set question, and ask them to annotate the pair as 0: No paraphrase relationship; 1: on similar topics, but different questions; 2: approximately the same question (an adaptation of the paraphrase evaluation of [125]). We take 300 validation set questions and ask three crowd-workers to rate them against retrieved training questions on this scale, and consider the label with majority rating. To improve quality, we manually verify their annotations.

Table 2.5 shows that \textbf{81\%} of validation set questions have at least one paraphrase in the training set, while \textbf{all} annotated questions have at least one topically similar question in the

\textsuperscript{14}The ELI5 demo from [106] also retrieves the top-1 similar training set question. Qualitatively, we found many validation examples had near-identical train paraphrases.

\textsuperscript{15}We pay workers 4 cents per question pair ($8-12 / hr). We only hire workers from USA, UK and Australia with a 95\% or higher approval rating and at least 1000 approved HITs.
<table>
<thead>
<tr>
<th>qns with at least one train set paraphrase</th>
<th>81%</th>
</tr>
</thead>
<tbody>
<tr>
<td>qns with at least one train set topically similar</td>
<td>100%</td>
</tr>
<tr>
<td>% of all pairs marked paraphrases</td>
<td>39.5%</td>
</tr>
<tr>
<td>% of all pairs marked topically similar</td>
<td>47.8%</td>
</tr>
<tr>
<td>% of all pairs marked as non-paraphrases</td>
<td>12.7%</td>
</tr>
</tbody>
</table>

Table 2.5: A human evaluation measuring the amount of overlap between validation set questions (qns) and retrieved questions from the training set.

training set, which indicates substantial training / validation overlap. The experiment had “fair agreement” with a Fleiss \( \kappa \) of 0.29 [61, 146].

As manually annotating question overlap can be expensive and time-consuming, we also experiment with automatic overlap detection methods. In particular, we use a RoBERTa-large binary classifier [165] fine-tuned on the Quora Question Paraphrase (QQP) dataset [100] from the GLUE benchmark [274]. For 43.6% of the ELI5 validation set, this classifier marked at least one retrieved question as a paraphrase (46% for the 300 questions we annotated). Qualitatively, we notice that this classifier often mis-classifies retrieved questions that are valid paraphrases but exhibit significant lexical or syntactic divergence. This observation, along with the smaller fraction of valid paraphrases in the QQP training set (37%), partially explains the gap between automatic & human evaluations.

**Using retrieved QA for generation:** Since ELI5 contains significant amount of overlap between the training and validation sets, a system can simply copy the answers of retrieved training set questions instead of actually doing generation. Table 2.6 shows that by using the longest answer within the top-\( K \) retrieved questions, we outperform two prior systems (RAG, BART + DPR) that use retrieval-augmented generation. As an upper bound, we also consider a system which uses the best possible answer to retrieved training set questions in terms of ROUGE-L (best top-\( K \) train answer). This system gets 28.5 ROUGE-L, outperforming all others.
**ELI5 performance on overlapping QA:** Finally, we measure the performance difference between validation questions that overlap with the training set vs. those that do not. In Appendix A.6 we notice large differences of 6.6 RPrec, 8.1 R@5 in retrieval performance favoring the overlap subset, but only a small generation score gain of 0.8 F1, 0.4 R-L (which may be misleading, Section 2.3.3).

**Takeaway (careful held-out curation):** Based on our findings, we suggest that more careful dataset curation for LFQA tasks is needed to prevent duplicates. We acknowledge the efforts from [57] to fix this issue, and suggest alternative methods to control overlap and focus on evaluating generalization in held-out sets: (1) retrieving paraphrases and running human validation to eliminate them; or (2) holding out entire genres or domains to reduce the possibility of overlap.

### 2.3.3 ROUGE-L Bounds on ELI5 Performance

We have seen that simply copying the answer of a close question paraphrase from the training set achieves 28.5 ROUGE-L with an optimal selection among retrieved questions and outperforming all computational models. But how “good” is this absolute number? What are some suitable upper & lower bounds to ROUGE-L scores on ELI5? Is ROUGE-L an informative metric for LFQA?

**Lower bounds** are trivial baselines used to test the vulnerability of datasets or metrics to simple heuristic strategies that do not actually perform the task. Recent examples include hypothesis-only baselines for natural language inference [79] and passage-only baselines for reading comprehension [115]. We evaluate two ROUGE-L lower bounds on ELI5: (1) copy the question 5 times and concatenate, as longer outputs boost ROUGE-L (Appendix A.8); (2) retrieve a *random* training set answer.
Table 2.6: Upper (↑) and lower (↓) bounds to performance on ELI5. Lower bounds have been submitted to the public KILT leaderboard, as “Metrics Test”.

Our first baseline contains entities often present in the gold answer, but without actually answering the question. Our second baseline follows the “style” of an answer but is completely off-topic.

As an upper bound, we estimate the ROUGE-L of gold answers themselves. On an average, there are 12 gold answers per question, so we measure the ROUGE-L of the longest gold answer with respect to the other gold answers. We also measure the maximum pairwise ROUGE-L between two gold answers for the same question.\(^{16}\) We only calculate upper bounds for the validation set, since the gold answers of the KILT test set are hidden.

**Lower bounds beat prior work, upper bounds have low ROUGE-L**: We compare our bounds with actual retrieval augmented generation systems in Table 2.6. Both our lower bounds (random training answer, copy input) are quite competitive, outperforming

\(^{16}\)Note that different gold answers were not written independently as Reddit users writing answers can read existing answers and may want to provide a non-overlapping perspective. Due to the high train/valid overlap, the best top-7 retrieved answer could be a better upper bound since it is from another Reddit post (and performs better than best gold answer).
RAG [155] and performing close to BART + DPR [200] without actually answering the question! This shows that ROUGE-L is fairly sensitive to simply copying entities from the question as well as stylistic properties of ELI5. On the other hand, upper bounds (longest gold answer) perform worse than our system (21.2 vs 24.4). Suspecting that this result is misleading, we run another human A/B test by showing volunteers a question and asking them to choose between answers generated by our system and the longest gold answer, shuffled at random. As seen in Table 2.3, the majority of humans prefer the gold reference answers vs generations (68% vs 14% for $p = 0.6$). In interviews with human annotators after completing the task, they reported that both answers were often fluent and stylistically similar, but one eventually veered off-topic.

**Takeaway (better automatic metrics needed):** Our experiments demonstrate that computing the ROUGE-L of generations against gold answers is not a meaningful way to evaluate LFQA systems, since it is not selective enough to differentiate between valid/invalid answers. There is a very small margin of improvement between trivial lower bounds and strong upper bounds, with the absolute scores of upper bounds being quite low. We suspect this is due to the long length of answers and fairly unconstrained and large output space. The ELI5 dataset has several open-ended questions with many plausible answers (like *What causes traffic*?), often involving analogies. A possible fix is a sentence-level evaluation and then aggregating scores across generated sentences, but appropriate penalties are needed for lack of diversity [318] and short lengths. Other possible fixes include learning task-specific metrics to measure semantic overlap [228] or metrics to check factual correctness [315] and faithfulness to input [272, 48, 317]. Ultimately, all automatic metrics have their limitations, and human evaluation is necessary [26].

---

17Human A/B testing details in Appendix A.7.
2.3.4 Difficulty of Human Evaluation

To better understand the inherent difficulty of evaluation in ELI5, we interviewed human annotators (of Table 2.3) and found two challenges:

(1) **Unfamiliarity with question topics**: Annotators were often unfamiliar with the technical topics discussed in some questions, which makes it hard to assess answer correctness. The ELI5 dataset has questions in a wide variety of topics (History, Politics, Biology etc.), while most annotators were CS graduate students. While we did allow annotators to use Wikipedia, they mentioned domain-experts will be better judges of answer quality.

(2) **Length of Answers**: Annotators mentioned the paragraph-long length of answers made the task quite challenging. Annotators reported taking an average of 2 minutes per answer pair, many of which required careful thought & concentration. This was especially difficult when only part of the answer was correct and the rest had contradictions or repetitions, a common theme in our generations.

**Takeaway**: Human evaluation is challenging but necessary for evaluating LFQA. Crowdworkers are unlikely to spend time reading & analyzing long text [5]. Hence, it is imperative to design *simpler* evaluations. One effort in this direction is [47], who reveal one generated sentence at a time and estimate system quality based on the number of sentences which fooled humans. Another promising direction is extrinsic evaluation [26] where humans actually interact with systems in real-world scenarios such as the Alexa Prize [214] or STORIUM [5].

2.4 Conclusion

In this chapter present a “retrieval augmented” generation system that achieves state of the art performance on the ELI5 long-form question answering dataset. However, an in-depth analysis reveals several issues not only with our model, but also with the ELI5
dataset & evaluation metrics. We hope that the community works towards solving these issues so that we can climb the right hills and make meaningful progress.

**Ethical Considerations**

Our system faces a similar set of issues as most modern text generation technology, like fabrication of facts [308], potential for misuse [23] and reflecting biases prevalent on Reddit (the ELI5 dataset has been built using the r/ELI5 subreddit). In our work, we attempted to make text generators more factually grounded by conditioning generations on retrieved Wikipedia articles, hoping to reduce fact fabrication. Unfortunately, a thorough analysis (Section 2.3.1) has revealed that our system is still not grounding its generations in retrievals, and we have recommended the design of better metrics to measure factual correctness to tackle this issue.

Our final models were trained using 64 Google Cloud TPUs for a total of 32 hours. As mentioned in the Google 2019 environment report, TPUs are highly efficient chips which have been specifically designed for machine learning applications”. These accelerators run on Google Cloud, which has “matched 100% of its electricity consumption with renewable energy purchases, and has committed to fully decarbonize its electricity supply by 2030” (https://cloud.google.com/sustainability). More details on training time are provided in Appendix A.2.

**Impact / Retrospective**

This study was originally published in NAACL 2021 [131]. Since then and up to August 2023, these findings have been used in over ninety five follow-up studies, and they have played an important role in the development of OpenAI’s WebGPT [186], which has inspired commercial retrieval-augmented systems like Perplexity AI and Bing Chat. Interestingly,
recent work has shown that the latest search-engine augmented large language models continue to suffer from hallucination issues [163], often producing text not fully supported in the underlying citation. Our work has also prompted researchers to create a new version of the ELI5 dataset which fixes train/validation overlap issues [20].
CHAPTER 3
RANKGEN: IMPROVING TEXT GENERATION WITH LARGE RANKING MODELS

Motivation and Summary: The model developed in Chapter 2 suffered from one major issue — its generations were often unrelated (“unfaithful”) to the retrieved documents which are supplied as inputs. This is a persistent issue even in modern large language models like GPT3.5 are augmented with search engines [163].

Can we make language models more faithful to their input? To tackle this issue, in this chapter I build RANKGen, a 1.2B parameter encoder model for English that scores model generated continuations to an input sequence (“prefix”). RANKGen can be flexibly incorporated as a scoring function in beam search and used to decode from any pretrained language model. I train RANKGen using large-scale contrastive learning to map a prefix close to the ground-truth sequence that follows it and far away from two types of negatives: (1) random sequences from the same document as the prefix, and (2) sequences generated from a large language model conditioned on the prefix. Experiments across four different language models (345M-11B parameters) and two domains show that RANKGen significantly outperforms decoding algorithms like nucleus, top-\(k\), and typical sampling on both automatic metrics (85.0 vs 77.3 MAUVE) as well as human evaluations with English writers (74.5% human preference over nucleus sampling). Analysis reveals that RANKGen outputs are more relevant to the prefix and improve continuity and coherence compared to baselines.\(^1\)

\(^1\)Resources accompanying this chapter can be found in https://github.com/martiansideofthemoon/rankgen.
3.1 Introduction

Despite exciting recent progress in large-scale language modeling [208, 23], text generated from these language models (LMs) continues to be riddled with artifacts. Modern LMs suffer from the “likelihood trap” [226, 309], in which high likelihood (low perplexity) sequences produced by greedy decoding or beam search tend to be dull and repetitive. While truncated sampling methods such as top-$k$ [58], nucleus [89], and typical sampling [176] alleviate these issues, they can also produce text with inconsistencies, hallucinations, factual errors, or commonsense issues [172, 46, 131].

Part of the problem is that LMs are trained using “teacher forcing”, where they are always given the ground-truth prefix\(^2\) and asked to predict the next token. At test-time, however, the prefix can contain model-generated text, allowing errors to propagate during decoding [15]. This issue, combined with the observation that LMs overly rely on local context [117, 242], contributes to the generation of sequences that break coherence or consistency within a larger discourse-level context [278].

To address this issue we present RANKGEN, a 1.2 billion parameter English encoder model that maps both human-written prefixes and model-generated continuations of those prefixes (generations) to a shared vector space. RANKGEN efficiently measures the compatibility between a given prefix and generations from any external LM by ranking the generations via their dot product with the prefix (Figure 3.2). We train RANKGEN using large-scale contrastive learning, encouraging prefixes to be closer to their gold continuation and far away from incorrect negatives. Since our objective considers two sequences rather than just single token prediction, it encourages RANKGEN to consider longer-distance relationships between the prefix and continuation rather than just local context.

We devise two different strategies (shown in Figure 3.1) for selecting challenging negative samples, and empirically show that current large LMs cannot distinguish gold

\(^2\)A prefix is a sequence of tokens fed as input to an LM, which then generates continuations conditioned on the prefix. A prefix is also called a prompt in prior work [58].
This financial set-back, while it had injured, for the time, Arthur Breen’s reputation for being “up and dressed,” had not, to any appreciable extent, curtailed his expenditures or narrowed the area of his social domain.

Mrs. Breen’s dinners and entertainments had been as frequent and as exclusive, and Miss Corinne had continued to run the gamut of the gayest and best patronized functions …

The drooping lips straightened and a shrewd, searching glance shot from Arthur Breen’s eyes. There was a brain behind this sleepy face—as many of his competitors knew …

He was still the patron of the night clubs; still he dined frequently with Garry O’Neil and Stephen O’Neil; still he appeared occasionally as a guest in the house where Garry O’Neil and Stephen O’Neil dined …

continuations from the negatives via perplexity (Section 3.2.1). In the first strategy, \textit{INBOOK}, we select random sequences that occur within the same document as the prefix. While these human-written negatives are fluent and might contain topic or entity overlap, they are irrelevant as continuations to the prefix. In the second strategy, \textit{GENERATIVE}, we generate continuations by conditioning a large pretrained LM on a given prefix. Compared to \textit{INBOOK} negatives, these negatives are much more relevant to the prefix, but they suffer from issues like hallucination and repetition.

While \textit{RANKGen} can be easily used to rerank full-length samples from any external LM, we demonstrate further improvements in generation quality when it is integrated as a scoring function into beam search. On automatic and human evaluations across four large pretrained models (345M to 11B parameters) and two datasets, we observe that \textit{RANKGen} significantly and consistently outperforms sampling-based methods (nucleus, typical, top-$k$) as well as perplexity-based reranking (85.0 vs 77.3 MAUVE, 74.5% human
preference over nucleus sampling). Additionally, RANKGen outperforms newer decoding algorithms like contrastive decoding and search (89.4 vs 84.9 MAUVE on Wikipedia) which were proposed after the initial RANKGen release in May 2022. Qualitative analysis from our human annotators (English writers) suggests that most of the improvements stem from increased relevance and continuity between the generated text and the prefix. Finally, we explore applications of our RANKGen retriever outside of text generation and report: (i) 3.4% pass@1 improvements over code-davinci-002 in Python text-to-code generation on the HumanEval benchmark; (ii) state-of-the-art results on two complex literary retrieval benchmarks: RELiC [247] and ChapterBreak [244]. We have open sourced our code and model checkpoints.

### 3.2 RANKGen: a generation ranker

RANKGen is a deep encoder network that projects prefixes and generations to a shared vector space. Given a prefix vector and a generation vector, we compute a score for the generation via the dot product between the two vectors. To ensure that these scores are meaningful, we train RANKGen using large-scale contrastive learning [207], pushing the prefix vector close to the gold completion and away from the vectors of negative samples (Figure 3.1). We use two types of negative samples for learning the metric space: (1) sequences at random locations in the same document (INBOOK), and (2) model generations (GENERATIVE). This section empirically justifies our negative sample choice (Section 3.2.1) before presenting a precise model formulation (Section 3.2.2).

---

3See Table 3.3, 3.4 for all results. MAUVE [203] is a recently introduced automatic metric for open-ended generation which has high correlation with human judgements.

4https://github.com/martiansideofthemoon/rankgen
Table 3.1: How often do models prefer the gold continuation to a prefix over an INBOOK negative (text from a different location in same document)? Overall, large LMs (via perplexity) perform poorly compared to both RANKGEN and humans. *GPT3 scores use 1000 data-points; †hard sets adversarially built with this model.

3.2.1 LMs do not choose gold over negatives

We explicitly choose our negatives to focus on a weakness of modern LMs which we empirically verify below: LMs often assign high probability to implausible or irrelevant continuations of a prefix.

**INBOOK negatives:** Our first type of negative samples are sequences from random locations in the same document as the prefix, whose lengths match those of the ground-truth continuations. As these negatives are written by humans, they are always fluent and coherent, and often topically similar to the prefix (with overlapping entities). However, they are irrelevant as continuations to the prefix, breaking discourse-level continuity and coherence [87, 74].

**LMs struggle to distinguish gold continuations from INBOOK negatives:** Given a prefix of 256 tokens from Wikipedia or a PG19 book [209], we measure how often LMs assign higher probability (lower perplexity) to the gold 128-token continuation over a single
<table>
<thead>
<tr>
<th>Discriminator</th>
<th>PG19</th>
<th>Wikipedia</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Unigram Overlap</td>
<td>40.2</td>
<td>44.4</td>
<td>42.3</td>
</tr>
<tr>
<td>GPT2-medium [208]</td>
<td>14.7</td>
<td>23.3</td>
<td>19.0</td>
</tr>
<tr>
<td>GPT2-XL [208]</td>
<td>21.5</td>
<td>31.5</td>
<td>26.5</td>
</tr>
<tr>
<td>T5-XXL (f.t. PG19)</td>
<td>32.4</td>
<td>33.7</td>
<td>33.1</td>
</tr>
<tr>
<td>T5-XXL-C4 [152]</td>
<td>19.0</td>
<td>39.1</td>
<td>29.1</td>
</tr>
<tr>
<td>RANKGEN (ours)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG-XL-GENERATIVE</td>
<td><strong>94.7</strong></td>
<td><strong>89.2</strong></td>
<td><strong>91.9</strong></td>
</tr>
<tr>
<td>PG-XL-INBOOK</td>
<td>69.8</td>
<td>59.7</td>
<td>64.8</td>
</tr>
<tr>
<td>PG-XL-both</td>
<td>92.0</td>
<td>74.9</td>
<td>83.5</td>
</tr>
<tr>
<td>all-XL-both</td>
<td>86.2</td>
<td>81.3</td>
<td>83.7</td>
</tr>
</tbody>
</table>

Table 3.2: How often do different models prefer the gold continuation to a prefix over a GENERATIVE negative (model-generated continuation)? LM perplexity strongly prefers GENERATIVE over gold continuations, while RANKGEN accurately prefers the gold. Negatives were generated from all four LM models in table using nucleus sampling [89] with \( p = 0.9 \) and then pooled (Appendix B.3.3 breaks down scores by LM).

We break all prefixes and continuations at sentence boundaries to make the task less reliant on local syntactic patterns. Table 3.1 shows that even large LMs perform far below human estimates on this task (63.2% for GPT2-XL vs 91.0% human on Wiki), and repeating this experiment with “hard” negatives selected from a trained RANKGEN model drops LM performance even further (50.7% for GPT2-XL vs. 90.5% human on Wiki). We hypothesize that LMs perform poorly because (1) they overly focus on local context instead of long-range dependencies from the prefix [117, 242]; and (2) LMs assign high likelihood to words with high frequency in their training data [91] which may occur in INBOOK but not in the gold continuation.

---

5We experiment with multiple INBOOK negatives in appendix §B.3.2. This task is similar to suffix identification tasks like ROCStories [185]; see §B.3.5 for experiments on them.

6Human study done on Upwork; details in Appendix B.2.

7See Appendix B.3.1 for more details on “hard negatives”.

34
LMs also struggle to distinguish gold continuations from **GENERATIVE** negatives:

Our second type of negative samples are continuations to a prefix that are generated by a pretrained LM. Machine-generated text is known to differ significantly from human text, containing repetitions, hallucinations, and artifacts [308, 175, 89]. We use these negatives to encourage **RANKGEN** to prefer generations closer to the human distribution, similar in spirit to GAN discriminators [67]. **GENERATIVE** negatives have also been used in previous energy-based LMs [38], although not at this scale; see Section 3.5 for more related work. In Table 3.2, we show that LM perplexity is poor at identifying human text over **GENERATIVE** negatives (GPT2-XL gets just 26.5% accuracy, well below 50% random chance). This relates to prior work showing LMs have high confidence in machine-generated text [65], especially their own (Appendix B.3.3).

### 3.2.2 Training RANKGEN

Having motivated our negative sampling strategies, we now describe **RANKGEN**’s training process. We train **RANKGEN** using large-scale contrastive learning with in-batch negative sampling, which is a popular metric learning technique [238] previously used for dense retrieval (DPR, [113]), image classification (SimCLR, [30]), and multimodal representation learning (CLIP, [207]).

A single **RANKGEN** training instance consists of a triple \((p_i, c_i, g_i)\), where \(p_i\) is a prefix, \(c_i\) is the ground-truth continuation of that prefix, and \(g_i\) is a continuation generated by an LM. We prepend a special token (\(\text{pre}\)) to each prefix, and \(\text{suf}\) (\(\text{suffix}\)) to each continuation and generation. We then pass each element of the triple through a shared Transformer encoder [262], projecting them to fixed-size vectors \((p_i, c_i, g_i)\) using the representation of the special token. To train this model, we use a contrastive objective that pushes the prefix vector \(p_i\) close to the gold continuation vector \(c_i\), but away from both the generation vector \(g_i\) as well as all other continuation vectors \(c_j\) in the same minibatch (“in-batch negative sampling”),
\[ Z(p_i) = \sum_{c_j \in B} \exp p_i \cdot c_j + \sum_{g_j \in B} \exp p_i \cdot g_j \]

\[ P(c_i|p_i) = \frac{\exp (p_i \cdot c_i)}{Z(p_i)} \]

\[ \text{loss} = -\sum_{(p_i,c_i) \in B} \log P(c_i|p_i) \]

where \( B \) is a minibatch. All minibatch elements are sampled from the same document, which provides the INBOOK negatives. Note that the minibatch size \( |B| \) is an important hyperparameter since it determines the number of negative samples; we set \( |B| = 1536 \) for our XL variant.\(^8\)

**Dataset construction:** We consider all possible 256-word prefixes \( p_i \) in our document, ensuring that prefixes begin and end at sentence boundaries. We then select continuations \( c_i \) of variable length (10-128 words long) for each prefix \( p_i \) so that RANKGEN can re-rank candidates of different lengths at test-time. To produce GENERATIVE negatives, we first use 50\% of our \((p_i, c_i)\) training data pairs to fine-tune T5-XXL [211] for causal language modeling (one per domain). For the remaining half of the dataset, we use this LM to generate a single continuation \( g_i \) to the prefix \( p_i \) of variable length (10-128 words) using nucleus sampling [89] with \( p = 0.9 \).

**3.2.3 Using RANKGEN at inference**

After model training, the dot product between the prefix and continuation vectors denotes their compatibility score. We experiment with two strategies for using these scores during generation: (1) over-generation and reranking, in which we use any pretrained LM and decoding algorithm to generate multiple samples (20 in our experiments) and then re-rank them; and (2) beam search (Figure 3.2), in which we generate \( N \) samples of length \( L \) via nucleus or ancestral sampling, compute the top \( B \) highest-scoring samples via RANKGEN, and concatenate them to the prefix to continue generation. There are three hyperparameters

---

\(^8\)See Appendix B.1.1 for training details and sizes of model variants.
Step 1: Given a prefix, generate $N$ samples $(s_1, ..., s_N)$ of length $L$ from a generator using any decoding algorithm.

Step 2: Score each sample based on its compatibility with prefix using RankGen.

Step 3: Take the top-$B$ samples (beam size $B$) and concatenate them to the prefix to continue generation.

Prefix: The winter had been dark and tedious. For some reason or other I had not been able to read much, and I began to think there were signs of the coming end. Suddenly, with hardly any warning, spring burst upon us. Day after day we had clear, warm sunshine which deepened every contrast of colour, and at intervals we were blessed with refreshing rains. I spent most of my time out of doors on the edge of a favourite wood.

Figure 3.2: The RANKGEN setup during inference. RANKGEN can be flexibly plugged into any generative model (like GPT2) using any decoding algorithm (like nucleus sampling) during inference in a beam-search like setup. The examples shown here are actual generations from GPT2-md (with nucleus $p=0.9$) and scores from RANKGEN.

for our beam search: (i) the rerank length $L$, or the number of tokens generated before each re-ranking; (ii) the beam size $B$; and (iii) the number of samples generated per beam $N$. Setting $N=20$, $B=1$, $L=128$ (max generation length) is equivalent to the first strategy of over-generation and re-ranking. Details of our implementation and hyperparameter search are in Appendix B.1.2, B.1.3. Overall all tested hyperparameters improve over baselines, but $N=10$, $B=2$, $L=20$ performs best but all tested hyperparameter choices improve over baselines (Figure 3.3).

3.3 Experiments

We compare RANKGEN to existing decoding algorithms (e.g., nucleus sampling, greedy decoding) across a wide range of pretrained LM decoders and RANKGEN configurations. In our experiments, we provide a human-written prefix to a pretrained LM and decode continuations from the model using RANKGEN as well as competing approaches. Our automatic evaluations measure the quality of the model-generated continuations using the MAUVE metric proposed by [203], on which RANKGEN outperforms all competing decoding ap-
proaches when used as a re-ranker of multiple samples. Furthermore, integrating RankGen into beam search yields the highest-quality outputs, measured both by MAUVE and human A/B tests. Finally, we also evaluate RankGen on Python text-to-code generation, and notice strong improvements over the OpenAI Codex (code-davinci-002) model. This section details our experimental setup and then highlights the main takeaways of our results.

3.3.1 Model configurations

**RankGen variants:** We study four configurations of RankGen, each with 1.2B parameters (XL size) and trained with minibatch size 1536. Three variants are trained on the PG19 dataset [209], which consists of long-form books, using (1) only INBOOK negatives, (2) only GENERATIVE negatives, and (3) both types of negatives. Since PG-19 contains mainly historical literature, we also experiment with different data sources by training RankGen on the union of four domains (“all”) — PG19, Wikipedia, C4-NewsLike and C4-WebTextLike [211]. This last model is trained using both types of negatives. More ablations varying the model size and minibatch size (number of negatives) are provided in Appendix B.5.

**Pretrained language models:** Does RankGen improve generation quality regardless of the size and pretraining dataset of the LM? To check this we evaluate four different pretrained LMs whose sizes vary considerably from that of RankGen (1.2B parameters). We experiment with two variants of GPT-2 [208]: GPT2-medium (345M) and GPT2-XL (1.5B parameters). We also evaluate a pretrained T5-XXL-v1.1 [211] model (11B parameters) that we fine-tune to perform language modeling on the training set of PG19 [209]. Finally, to experiment with a large LM trained on out-of-domain data for RankGen-PG19, we evaluate the T5-XXL model from [152] (11B parameters) that was fine-tuned for language modeling on the C4 corpus.
3.3.2 Open-ended text generation

Following prior work on text generation [285, 89, 241], we primarily focus on open-ended text generation, which has wide applications for tasks such as generating stories [58], poetry [314], and dialog [181] and few-shot NLP [23]. We consider two domains in our study: (1) prefixes from Wikipedia, and (2) literary text from PG19 [209]. Since it is difficult to conduct human evaluations of long sequences of machine-generated text [112], our main experiments consider a 256-token prefix and 128-token generations. We analyze generation quality given varying prefix lengths in Section 3.4.3.

Decoding algorithms: For each LM considered we decode outputs using greedy decoding, ancestral sampling, nucleus sampling [89], top-k sampling [58], and typical sampling [176]. Since RankGen is fundamentally a re-ranker of multiple samples, we also compare to two other re-rankers using LM perplexity and unigram overlap, respectively. In all re-ranking settings, we generate 20 samples and then re-rank them with each method. For RankGen, we also use beam search (§3.2.3) that re-ranks partially generated hypotheses.

Automatic & human evaluation metrics: We use MAUVE [203] as our primary metric for automatic evaluation. MAUVE computes the similarity of the distribution of human-written text and machine-generated text, and has high correlation with human judgments. Since automatic metrics are insufficient for text generation evaluation [26], we also conduct a human evaluation by hiring English teachers and writers from Upwork; see Appendix B.2 for more details. For each of GPT2-medium and T5-XXL-C4 we choose 50 Wikipedia and 50 PG19 prefixes, and show three annotators a pair of continuations from different decoding strategies in a random order (blind A/B testing). Annotators are asked to choose the better continuation and provide a 1-3 sentence explanation for their choice. This gives us 600 annotations, analyzed in §3.3.4, 3.4.1.

9Details about our MAUVE setup in Appendix B.4.1. More evaluations with metrics like REP [286] in Appendix B.4.3.

10https://www.upwork.com
Table 3.3: A comparison between RANKGEN variants and baseline decoding algorithms using MAUVE [203], an automatic text generation metric with high human correlation. RANKGEN significantly outperforms baselines like nucleus & typical sampling, as well as other re-ranking strategies using LM perplexity and unigram overlap. Incorporating RANKGEN into beam search (last row) results in the best average MAUVE score.

### Results from automatic evaluations

Table 3.3 contains MAUVE scores for all decoding configurations and datasets. Overall, we see that:

**RANKGEN re-ranking and beam search significantly improves MAUVE:** Re-ranking full-length samples with RANKGEN yields an average MAUVE score of 83.4 across all configurations, significantly outperforming other decoding strategies like greedy decoding (15.4), ancestral sampling (74.8), and nucleus / top-k / typical sampling (77.1-77.4). Adding beam search further boosts performance to 85.0.\(^{11}\) Surprisingly, re-ranking 20 full-length ancestral samples with RANKGEN performs better than standard nucleus sampling (77.3 vs 82.6). However, re-ranking 20 ancestral samples is slightly worse than re-ranking 20 nucleus samples (82.6 vs 83.4) due to worse inherent quality of ancestral vs nucleus (74.8 vs 77.3).

---

\(^{11}\)Hyperparameter grid search details in Appendix B.1.3.

---

<table>
<thead>
<tr>
<th>Decoding method</th>
<th>T5-XXL-C4 PG19 wiki</th>
<th>GPT2-md PG19 wiki</th>
<th>GPT2-XL PG19 wiki</th>
<th>T5-XXL-PG19 PG19 wiki</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Greedy decoding</td>
<td>6.6</td>
<td>15.2</td>
<td>3.8</td>
<td>11.2</td>
<td>23.4</td>
</tr>
<tr>
<td>Ancestral sampling</td>
<td>67.7</td>
<td>71.6</td>
<td>75.5</td>
<td>73.2</td>
<td>77.4</td>
</tr>
<tr>
<td>Nucleus, (p = 0.9) [89]</td>
<td>69.7</td>
<td>77.9</td>
<td>73.0</td>
<td>74.6</td>
<td>74.4</td>
</tr>
<tr>
<td>Top-k, (k = 40) [58]</td>
<td>68.3</td>
<td>77.3</td>
<td>74.8</td>
<td>73.4</td>
<td>76.0</td>
</tr>
<tr>
<td>Typical, (p = 0.9) [176]</td>
<td>69.5</td>
<td>77.4</td>
<td>73.2</td>
<td>73.5</td>
<td>73.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Re-ranking 20 full-length ancestral samples</th>
<th>RANKGEN PG19-XL-both</th>
<th>RANKGEN PG19-XL-I NB</th>
<th>RANKGEN PG19-XL-both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram overlap</td>
<td>65.6</td>
<td>80.7</td>
<td>74.8</td>
</tr>
<tr>
<td>LM perplexity</td>
<td>62.6</td>
<td>55.1</td>
<td>55.5</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-GENERATIVE</td>
<td>78.3</td>
<td>82.4</td>
<td>76.2</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-INBOOK</td>
<td>70.7</td>
<td>83.4</td>
<td>76.7</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-both</td>
<td>80.7</td>
<td>86.4</td>
<td>76.3</td>
</tr>
<tr>
<td>RANKGEN all-XL-both</td>
<td>73.0</td>
<td>88.1</td>
<td>74.8</td>
</tr>
<tr>
<td>+ beam search ((B=2, L=20, N=10))</td>
<td>74.0</td>
<td>89.4</td>
<td>76.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Re-ranking 20 full-length nucleus samples</th>
<th>RANKGEN PG19-XL-both</th>
<th>RANKGEN PG19-XL-I NB</th>
<th>RANKGEN PG19-XL-both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram overlap</td>
<td>65.6</td>
<td>80.7</td>
<td>74.8</td>
</tr>
<tr>
<td>LM perplexity</td>
<td>62.6</td>
<td>55.1</td>
<td>55.5</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-GENERATIVE</td>
<td>78.3</td>
<td>82.4</td>
<td>76.2</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-INBOOK</td>
<td>70.7</td>
<td>83.4</td>
<td>76.7</td>
</tr>
<tr>
<td>RANKGEN PG19-XL-both</td>
<td>80.7</td>
<td>86.4</td>
<td>76.3</td>
</tr>
<tr>
<td>RANKGEN all-XL-both</td>
<td>73.0</td>
<td>88.1</td>
<td>74.8</td>
</tr>
<tr>
<td>+ beam search ((B=2, L=20, N=10))</td>
<td>74.0</td>
<td>89.4</td>
<td>76.2</td>
</tr>
</tbody>
</table>
Table 3.4: Percentage of instances for which English writers prefer RANKGEN outputs over nucleus samples in a blind A/B test. Scores shown are majority vote, with mean accuracy in subscript. Humans significantly prefer RANKGEN ($p < 10^{-3}$); agreement stats in Table 3.6.

<table>
<thead>
<tr>
<th>Decoding Method</th>
<th>PG19</th>
<th>Wikipedia</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT2-md</td>
<td>80.0 (72.0)</td>
<td>82.0 (78.3)</td>
<td>81.0 (75.1)</td>
</tr>
<tr>
<td>T5-XXL-C4</td>
<td>68.0 (63.3)</td>
<td>68.0 (65.3)</td>
<td>68.0 (64.3)</td>
</tr>
<tr>
<td>Overall</td>
<td>74.0 (67.8)</td>
<td>75.0 (71.9)</td>
<td>74.5 (69.8)</td>
</tr>
</tbody>
</table>

Table 3.5: A comparison of RANKGEN with newer decoding methods proposed after the initial RANKGEN release (May 2022). RANKGEN outperforms all methods in terms of MAUVE scores [203].

<table>
<thead>
<tr>
<th>Decoding Method</th>
<th>GPT2-md PG19 wiki</th>
<th>GPT2-XL PG19 wiki</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nucleus ($p = 0.9$)</td>
<td>73.0 74.6</td>
<td>74.4 75.0</td>
</tr>
<tr>
<td>Eta [85]</td>
<td>76.4 72.8</td>
<td>77.7 76.2</td>
</tr>
<tr>
<td><strong>Contrastive methods</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>search [241]</td>
<td>5.3 21.2</td>
<td>54.0 43.2</td>
</tr>
<tr>
<td>decode [158]</td>
<td>65.2 83.2</td>
<td>73.2 84.9</td>
</tr>
<tr>
<td>RANKGEN-all-XL (ours)</td>
<td><strong>79.0</strong> 84.9</td>
<td><strong>79.0</strong> 86.4</td>
</tr>
<tr>
<td>rerank full ancestral</td>
<td></td>
<td></td>
</tr>
<tr>
<td>beam search nucleus</td>
<td>76.2 <strong>88.9</strong></td>
<td>77.0 <strong>89.4</strong></td>
</tr>
</tbody>
</table>

Re-ranking generations by unigram overlap to the prefix is a surprisingly good baseline (79.7), while re-ranking by LM perplexity reduces MAUVE to 65.2, since it emulates likelihood-based methods like greedy decoding. Finally, RANKGEN performs best on in-domain data, with the PG19-XL-both variant obtaining better scores than the model trained on four domains (80.7 vs 73.0 on T5-XXL-C4, PG19).

**INBOOK negatives help more than GENERATIVE, but using both maximizes MAUVE:**

In Table 3.3 (bottom), we perform ablations by removing the INBOOK and GENERATIVE for RANKGEN PG19 variants. All three variants outperform nucleus sampling (77.3), but keeping both objectives performs best (82.6). A model trained with only INBOOK is more effective (81.4) than one trained with only GENERATIVE (80.2).
**RANKGEN outperforms newer decoding algorithms proposed after RANKGEN release:**

Since the release of RANKGEN in May 2022, several new decoding algorithms have been proposed including contrastive search [241, 240], contrastive decoding [158], and eta sampling [85]. In Table 3.5, we compare RANKGEN to these newer methods on GPT2-md and GPT2-XL. Overall, we find that RANKGEN significantly outperforms all newly proposed decoding algorithms (89.4 vs 84.9 on GPT2-XL wikipedia against the best baseline contrastive decoding).

### 3.3.4 Human evaluation with A/B tests

Despite the high human correlation of MAUVE, human evaluation remains critical for open-ended generation [26, 64]. Since human evaluation is expensive, we focus on comparing our best performing RANKGEN variant (RANKGEN-XL-all with beam search) to nucleus sampling, one of the most popular decoding algorithms in use today. We conduct blind A/B testing comparing the two methods, hiring English teachers and writers on Upwork (§3.3.2). Table 3.4 shows that humans significantly prefer outputs from RANKGEN over nucleus sampling (74.5% preference by majority vote, \( p < 0.001 \)). RANKGEN preference is higher with more inter-annotator agreement (Table 3.6) for outputs from the smaller GPT2-medium. Finally, humans show slightly higher RANKGEN preference for Wikipedia generations compared to PG19.

### 3.3.5 Text to Python code generation

In the experiments so far, we primarily focused on open-ended text generation. How well does RANKGEN work on other long-form generation tasks? In this section, we evaluate RANKGEN on Python source code generation on the OpenAI HumanEval benchmark [27]. Given a docstring as input, the objective of the HumanEval task is to produce a Python

---

12We use the official implementations for all these methods. Links - contrastive search, contrastive decoding, eta sampling.
Table 3.6: Inter-annotator agreement for the human evaluation in Table 3.4 using Fleiss $\kappa$ [61], and % of pairs with unanimous agreement among 3 annotators. Overall we see moderate agreement, higher for Wiki, GPT2.

<table>
<thead>
<tr>
<th>Model</th>
<th>PG19</th>
<th>Wikipedia</th>
<th>Overall</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT2-md</td>
<td>0.31, 48%</td>
<td>0.49, 60%</td>
<td>0.40, 54%</td>
</tr>
<tr>
<td>T5-XXL-C4</td>
<td>0.27, 46%</td>
<td>0.30, 48%</td>
<td>0.29, 47%</td>
</tr>
<tr>
<td>Overall</td>
<td>0.29, 47%</td>
<td>0.40, 54%</td>
<td>0.35, 51%</td>
</tr>
</tbody>
</table>

Table 3.7: Performance of RANKGEN on text-to-code generation on the OpenAI HumanEval benchmark [27]. Re-ranking Codex outputs using RANKGEN improves the pass@1 by 3.4%.

<table>
<thead>
<tr>
<th>Model</th>
<th>pass@1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random guessing</td>
<td>0.0</td>
</tr>
<tr>
<td>PaLM coder [31]</td>
<td>36.0</td>
</tr>
<tr>
<td>OpenAI Codex (code-davinci-002) [27]</td>
<td>49.1</td>
</tr>
<tr>
<td>+ RANKGEN code-large-INBOOK (ours)</td>
<td>52.5</td>
</tr>
</tbody>
</table>

function body solving the task specified in the docstring. Evaluation is automatic, using unit testing of the generated functions.

**Experimental Setup:** We first train a RANKGEN by fine-tuning the encoder of CodeT5-large [148] on Python source code extracted from the BigQuery Github corpus\(^\text{13}\). We train this model on INBOOK negatives only, and use docstrings as the prefix and the function body as the suffix to make the pretraining objective closer to text-to-code generation. To make the task more challenging, we replace every variable in the docstring and function body with placeholder tokens like `var1, var2, ...` and construct minibatches in a way such that each function has the same number of variables. This ensures the consistency

\(^{13}\text{https://console.cloud.google.com/marketplace/details/github/github-repos}\)
of variable across different functions in a minibatch, preventing the task from turning into a trivial variable set matching problem.

**Results:** We apply our RANKGEN-code model to re-rank Python code generations by the OpenAI Codex (code-davinci-002) model, and present our results in Table 3.7. Overall, we find that re-ranking RANKGEN significantly outperforms the base Codex model by 3.4% pass@1, showcasing the benefit of RANKGEN in domains outside text.

### 3.4 Analysis

Our experiments confirm that continuations decoded using RANKGEN are of higher quality than those generated via other methods. In this section, we explore what properties of the generated text improve with RANKGEN, examine its speed-quality trade-off, generalization to different prefix lengths, and utility as a retriever.

#### 3.4.1 In what ways does RANKGEN improve generation quality?

To get more insight into the human preference judgments made in Section 3.3.4, we asked our annotators to provide a 1-3 sentence free-form explanation for each of their choices. We manually categorized each of 600 explanations into nine broad categories loosely based on the SCARECROW schema designed by [46]. In Table 3.8 we see that 81% of the explanations preferring RANKGEN mentioned some aspect of the relationship between the prefix and the generated text, including relevance, continuity, and stylistic similarity. 8.0% of the explanations said that RANKGEN outputs displayed fewer commonsense errors, while 4.7% said that they were less repetitive. We show some generations and human explanations in Table 3.9 and several more full-length generations in Appendix B.6.

We conduct several ablation experiments in Appendix B.5 showing the performance tradeoffs when varying the model size and minibatch size (number of negative samples). Overall, we find that increasing model and minibatch size improves performance.

---

14 All 600 human explanations are provided in submission.
Table 3.8: Distribution of reasons given by our human evaluators (English writers/teachers) for preferring RANKGEN outputs over nucleus samples. Relevance / continuity to prefix was a common explanation.

### 3.4.2 How fast is decoding with RANKGEN?

Our algorithm requires over-generation followed by RANKGEN re-ranking. How much extra decoding time does this add? In Figure 3.3, we show the trade-off between MAUVE score and decoding time across different hyperparameters.\(^{15}\) While decoding a single nucleus sample takes just 0.8 seconds, generating 20 samples followed by re-ranking with RANKGEN requires 2.5 seconds. The best-performing hyperparameters use multiple re-ranking steps, taking 5.9 seconds.\(^{16}\) In Appendix B.1.3.2, we see that over-generation is the bottleneck, since re-ranking takes only a fraction of the time (1-10\%) compared to generation. Developing methods that avoid over-generation (e.g., via distillation) is an exciting future work direction.

\(^{15}\)Timing depends on library / hardware. We analyze HuggingFace on RTX3090, T5X on TPU-v3 in appendix B.1.2.

\(^{16}\)See Appendix B.1.3.2 for more speed tradeoff plots.
**Prefix Generations Annotator Preference**

**PG19. Half a Life-time Ago, by Elizabeth Gaskell:** ... If thou doesn’t choose to marry me on those terms–why! I can snap my fingers at thee, never fear. I’m not so far gone in love as that. But I will not have thee, if thou say’st in such a hectoring way that Willie must go out of the house–and the house his own too–before thou’lt set foot in it. ... “Thou hast may-be spoken a word too much,” said Michael, pale with rage.

Text 1 (Nucleus): “How on earth could it be? He must be part of the marriage, my love. But he can’t go away—he can’t go away with me. And if he does go away, he’ll have to pay ...

Text 2 (RANKGEN): “Take heed how thou handlest thy words in the future.” “I haven’t spoken a word too much,” she murmured, coolly. “I meant–I meant the last word!”

**Wiki, American Indian Movement:** ... named three lower-level AIM members involved in her death: Arlo Looking Cloud, John Graham, and Theda Nelson Clarke. This was the first time that an AIM leader active at the time of the Aquash murder had publicly accused AIM of having been involved. Looking Cloud and Graham were convicted of murder in 2004 and 2010, by ... By then Clark was being cared for in a nursing home and was not indicted.

Text 1 (Nucleus): Clarke was the second AIM leader to be charged. On July 19, 2009, her defense attorney, Michael Kranz had filed a motion ... His request for a new trial failed in December 2009. In 2009, the ...

Text 2 (RANKGEN): Clarke has also denied any involvement in Aquash’s murder. In the early months of 2001 the FBI began an effort to break through AIM’s defenses, to try to identify and bring charges against all three AIM members...

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Generations</th>
<th>Annotator Preference</th>
</tr>
</thead>
<tbody>
<tr>
<td>PG19. Half a Life-time Ago, by Elizabeth Gaskell: ... If thou doesn’t choose to marry me on those terms–why! I can snap my fingers at thee, never fear. I’m not so far gone in love as that. But I will not have thee, if thou say’st in such a hectoring way that Willie must go out of the house–and the house his own too–before thou’lt set foot in it. ... “Thou hast may-be spoken a word too much,” said Michael, pale with rage.</td>
<td>Text 1 (Nucleus): “How on earth could it be? He must be part of the marriage, my love. But he can’t go away—he can’t go away with me. And if he does go away, he’ll have to pay ... Text 2 (RANKGEN): “Take heed how thou handlest thy words in the future.” “I haven’t spoken a word too much,” she murmured, coolly. “I meant–I meant the last word!” “Swear it, thou hastna sworn a word yet...</td>
<td>Text 2. Text 1 has a completely different style, way more modern. First sentence in Text 2 fits the tone of enraged Michael. Text 2. The writing style is more similar, Text 1 sounds too modern. Plus, the atmosphere of the fragment is more consistent. The characters seem to be having an argument, so &quot;My love&quot; in doesn’t make sense.</td>
</tr>
</tbody>
</table>

| Wiki, American Indian Movement: ... named three lower-level AIM members involved in her death: Arlo Looking Cloud, John Graham, and Theda Nelson Clarke. This was the first time that an AIM leader active at the time of the Aquash murder had publicly accused AIM of having been involved. Looking Cloud and Graham were convicted of murder in 2004 and 2010, by ... By then Clark was being cared for in a nursing home and was not indicted. | Text 1 (Nucleus): Clarke was the second AIM leader to be charged. On July 19, 2009, her defense attorney, Michael Kranz had filed a motion ... His request for a new trial failed in December 2009. In 2009, the ... Text 2 (RANKGEN): Clarke has also denied any involvement in Aquash’s murder. In the early months of 2001 the FBI began an effort to break through AIM’s defenses, to try to identify and bring charges against all three AIM members... | Text 2 - The last sentence of the prefix paragraph ("By then Clark was being cared for in a nursing home and was not indicted") flows well with Text 2, implying that Clarke was absolved of guilt. Text 2. Text 2 further goes into Clark’s involvement in the case and Aquash’s murder while Text 1 contradicts part of the prefix. |

Table 3.9: Representative model outputs using RANKGEN vs nucleus sampling [89], along with human explanations (from English teachers/writers) for preferring RANKGEN. For every row the color coding grounds the annotator explanation in the prefix and generation. See Appendix B.6 for more full-length generations.

### 3.4.3 Generation with different length prefixes

Our RANKGEN model is trained with a fixed prefix length of 256 tokens, and all of the evaluations in Section 3.3 also assume a prefix length of 256 tokens. However, many text generation applications take shorter prefixes as input, like short writing prompts in story generation [58]. How well does RANKGEN generalize to shorter and longer prefixes? Figure 3.4 compares nucleus sampling to RANKGEN across varying prefix lengths. We observe that RANKGEN consistently outperforms nucleus sampling in terms of MAUVE,
Figure 3.3: Performance/time trade-off across hyperparameters (grid search details in §B.1.3). RANKGen re-ranking significantly improves MAUVE, but need an order of magnitude more time due to overgeneration.

Figure 3.4: MAUVE score variation with change in prefix length for GPT2-medium on Wikipedia. Across prefix lengths re-ranking with RANKGen-XL-all boosts performance, and using it in beam search does best.

and beam search with RANKGen always provides further gains, suggesting robustness to the prefix length.
3.4.4 **RANKGEN as a retriever**

While we designed RANKGEN for text generation, we find that it is also an effective zero-shot retriever. RANKGEN follows a dual encoder architecture similar to those of several recent dense retrievers like DPR [113] and REALM [81]. We test RANKGEN on RELiC [247], a complex literary retrieval task. Given a literary analysis excerpt, systems must retrieve a quote from a book which is most relevant to the excerpt. RELiC requires a deep understanding of literary phenomena (like irony, metaphors, co-reference, style), and current retrievers struggle on it. We test models in a zero-shot setting, without finetuning on RELiC training data. In Table 3.10 we find RANKGEN significantly outperforms other retrievers, achieving a new state of the art on RELiC. PG-XL-INBOOK performs best (6.0 vs 2.9 recall@1 against the next-best ColBERT), approaching a fully supervised upperbound (9.4). While our XL model has many more parameters than baselines, even the smaller-sized PG-base-both outperforms all baselines (3.8 vs 2.9). Dropping INBOOK leads to poor performance (0.7), further confirming its efficacy. Besides RELiC, we investigate retrieval over PG19 books in appendix §B.3.2, and suffix identification in §B.3.5, achieving state-of-the-art on ChapterBreak [244].

3.5 **Related Work**

Our work on RANKGEN draws inspiration from previous research on self-supervised learning, energy-based models, and modeling non-local dependencies. For instance, our INBOOK negative sampling is related to popular self-supervised representation learning methods that leverage discourse information across multiple sentences, which is useful for learning sentence embeddings [123, 86, 107]. Our formulation is most similar to QuickThought [168], which uses in-batch negative sampling on a contiguous set of sentences. More recently, the next sentence prediction task has been used for pretraining large LMs [42, 17].

---

17Anonymous Razorbill” on the official leaderboard.
Table 3.10: Performance on RELiC [247] compared to other retrievers. We achieve state-of-the-art on the zero-shot setting, nearing the supervised upperbound (†).

145, 8]. Unlike these works, we focus specifically on text generation rather than self-supervised pretraining for natural language understanding tasks.

RANKGen is also closely related to efforts in energy-based methods [149] for generative modeling [75, 198], speech recognition [275], open-ended text generation [12, 38], machine translation [234, 150, 19], constrained generation [206, 183], and models for specific attributes like style [36, 302], length [157], or repetition & relevance [90]. Unlike prior work, we use human-written text from the same document as negative samples (INBOOK) in addition to machine-generated text. RANKGen is also trained at a much larger scale than prior energy-based models for text (1.2B parameters, contrastive learning with 3K negatives on 4 domains).

Finally, RANKGen is related to efforts in modeling non-local dependencies in generation, which include methods that predict multiple tokens [192, 205], rely on retrieval [118], use bidirectional LMs [229], employ contrastive learning [241, 6], use BERT for sentence-level language modeling [99], and designing sequence-level losses [290, 51, 286, 167] for reducing exposure bias [15, 216]. While the RANKGen approach is significantly different
from these prior works, it can be intuitively viewed as a “k-word sequence-level” language modeling approach, which is discriminative rather than generative.

3.6 Conclusion and Future Work

We present RANKGEN, a large encoder which scores continuations given a prefix and can be plugged into any text generation system. RANKGEN significantly outperforms popular decoding methods on both automatic and human evaluations. We note several exciting future directions for RANKGEN, including:

- training (or adapting) a multilingual variant of RANKGEN, as our current RANKGEN models are trained on English text only
- training larger RANKGEN models (T5-XXL size or bigger), with longer prefix / suffix lengths, to see if generation quality continues to improve with scale
- exploring the utility of RANKGEN in other generation tasks like dialog generation, summarization, or long-form question answering
- RANKGEN re-ranking of significantly larger hypothesis sets generated using search algorithms like that in [300]
- more directly incorporating RANKGEN into generative modeling to eliminate the need for over-generation, either via gradient-based sampling [206], distilling RANKGEN knowledge into LMs via unlikelihood training [286] or reward modeling with RL [196]
- using RANKGEN as a retriever in knowledge retrieval augmented generation [186, 126]
- further exploring the capability of RANKGEN as a retriever, either zero-shot or by fine-tuning on retrieval benchmarks like BEIR [249]
- utilizing of RANKGEN as a text generation evaluation metric like CARP [173] or CLIPScore [84]
• using RankGen on other domains with sequential data, like code completion, protein synthesis, or generating mathematical proofs.

**Limitations**

An important limitation of RankGen compared to other decoding methods is the need for over-generation, which we discuss in Section 3.4.2. While RankGen itself is efficient, generating multiple samples increases decoding time by an order of magnitude. RankGen is a re-ranking method, so it relies on other decoding methods to produce the candidate output set. Biases in the output candidate set from existing decoding algorithms may be present in RankGen outputs. Besides this, RankGen may be vulnerable to adversarial examples [245] — gibberish text which gets high RankGen score, obtained by white-box attacks [50, 270].

This study is limited to open-ended text generation, which has a large space of possible outputs. RankGen or our findings may not be directly applicable to other generation tasks which have a more constrained output space like summarization, long-form QA or machine translation. However, we are optimistic about the generality of RankGen given the strong results in text-to-code generation.

**Impact / Retrospective**

This study was originally published in EMNLP 2022 [129]. Since then and up to August 2023, these findings have been used in over twenty five follow-up studies, and our PIP package has been downloaded over two thousand times.\(^\text{18}\). Besides text generation, RankGen has been used as an evaluator for open-ended text generation [279, 132], evaluator for long-form question answering [299], as well as for AI-generated text detection [132, 259].

\(^\text{18}\)https://pepy.tech/project/rankgen
An important point to note is the relationship between RankGen and reinforcement learning with human feedback (RLHF) [196], a technique that has gained immense popularity in the last one year due to the success of ChatGPT [225]. Both methods were developed in parallel, and achieve a similar goal of better aligning LM generations with their prefix. One advantage of RankGen over RLHF is that RankGen is a self-supervised method that can leverage web data, while RLHF needs a large and expensive human-annotated dataset. However, RLHF has some important advantages over RankGen: (1) RLHF goes a step further than RankGen, and teaches the base language model the knowledge in the reward model through RL; (2) The reward model in RLHF is exactly aligned to improving the test-time distribution of prefixes / prompts, allowing it to learn many other tasks besides open-ended generation. Utilizing RankGen as a reward model for RLHF, and designing self-supervised RankGen training data for custom downstream tasks (like reducing toxicity, instruction following), are exciting directions for future research.
CHAPTER 4
LONGEVAL: GUIDELINES FOR HUMAN EVALUATION OF FAITHFULNESS IN LONG-FORM SUMMARIZATION

Motivation and Summary: One of the key challenges in both Chapter 2 and Chapter 3 was the difficulty in evaluating long-form generated text. As discussed in those chapters, while human evaluation remains best practice for accurately judging the faithfulness of generated text, few solutions exist to address the increased difficulty and workload when evaluating long-form outputs.

In this chapter, I first shed light on current human evaluation practices surrounding long-form text generation, through a survey of 162 papers on long-form summarization. I find that 73% of these papers do not perform any human evaluation on model-generated summaries, while other works face new difficulties that manifest when dealing with long documents (e.g., low inter-annotator agreement). Motivated by this survey, we present LONGEVAL, a set of guidelines for human evaluation of faithfulness in long-form summaries that addresses the following challenges: (1) How can we achieve high inter-annotator agreement on faithfulness scores? (2) How can we minimize annotator workload while maintaining accurate faithfulness scores? and (3) Do humans benefit from automated alignment between summary and source snippets? We deploy LONGEVAL in annotation studies on two long-form summarization datasets in different domains (SQuALITY and PubMed), and we find that switching to a finer granularity of judgment (e.g., clause-level) reduces inter-annotator variance in faithfulness scores (e.g., std-dev from 18.5 to 6.8). We also show that scores from a partial annotation of fine-grained units highly correlates with scores from a full
annotation workload (0.89 Kendall’s $\tau$ using 50% judgments). I have also released this study’s human judgments, annotation templates, and software for future research.\footnote{https://github.com/martiansideofthemoon/longeval-summarization}

4.1 Introduction

Human judgments are considered the gold standard for evaluating model-generated summaries [136, 55] and generated text more broadly [26]. Unfortunately, human evaluation tends to be labor-intensive, expensive to scale, and difficult to design. This is problematic as a large number of judged examples is needed to draw statistically significant conclusions about system performances [284] or correlations between human judgments and automatic metrics [39]. Human evaluation is especially challenging when long sequences of generated text need to be evaluated, due to the inherent subjectivity in the task [112, 32, 131, 71].

To better understand the challenges of human evaluation on long-form summaries (150 words or longer), we first conduct a comprehensive survey of 162 publications and preprints on long-form summarization (Section 4.2). We find that 119 papers (73%) do not perform human evaluation on long-form summaries, while the remaining papers deviate significantly from suggested best practices for reproducibility [64]. Current human evaluation setups lack standardization in their design decisions (such as annotation granularity), some of which can significantly impact inter-annotator agreement (Section 4.3.1). Finally, 20 papers explicitly mention human evaluation is expensive, difficult, and time-consuming due to the long length of summaries and source documents.

To move towards a more consistent and efficient human evaluation, we present LONGEVAL, a set of guidelines for human evaluation of faithfulness in long-form summarization (Section 4.3). We empirically evaluate LONGEVAL using human annotation studies on two long-form summarization datasets: SQuALITY [273] and PubMed [33]. We provide an
Q1: Can inter-annotator agreement be improved with fine-grained annotations?

Q2: Can annotator workload be reduced by annotating just a fraction of the long summary?

Q3: Is it helpful to automatically align summary units with the long source document?

Asa Graybar is a biological engineer who studies keeping Slider eggs alive and he is accused of a crime at the opening of the story. He thinks he was framed by Tom Dorr, Hazeltyne’s general manager. He was offered one year as a “changeling” on another planet or 5 years in rehabilitation on Earth. He elects to do the one year, and thinks that he will get into smuggling Slider eggs on Jordan’s planet.....

Finding: Annotating faithfulness of individual summary clauses and aggregating them leads to significantly higher inter-annotator agreement, compared to the dominant paradigm of evaluating whole summaries at once via Likert ratings (std-dev 18.5 to 6.8 on SQuALITY).

RQ1: Can inter-annotator agreement be improved while evaluating faithfulness of long-form summaries via fine-grained annotations?

Finding: Despite annotating a fraction of summary clauses, faithfulness scores under a reduced workload maintain high correlation with those from a full workload (0.89 Kendall’s τ at 50% workload).

Figure 4.1: Overview of research questions considered in LONGEVAL. Example summary taken from SQuALITY.
RQ3: Do humans benefit from automatically aligning summary units to relevant sentences in the source document?

Finding: Unlike suggestions in prior work on short-form summarization [82, 137], aligning parts of the summary to source document is only useful when the summary is highly extractive or mostly correct.

Overall, our contributions are:

(1) a 162-paper survey of current human evaluation practices in long-form summarization;
(2) LONGEVAL, a set of three guidelines for evaluating faithfulness in long-form summarization;
(3) an empirical validation of LONGEVAL guidelines on two long-form summarization datasets in different domains (SQuALITY and PubMed);
(4) A dataset with 3-way fine-grained human faithfulness judgments for 120 SQuALITY & PubMed summaries annotated using LONGEVAL which can be used for benchmarking automatic metrics.

We open-source our human evaluation data, annotation interface, and code for future research.¹

4.2 Survey of human evaluation practices

Before discussing LONGEVAL, we first attempt to understand current human evaluation practices in long-form summarization through a comprehensive survey of 162 papers. Our survey reveals several concerning trends: absence of human evaluation, non-reproducible experimental setups, lack of standardization, and complaints of long summaries being challenging and expensive to evaluate. These results show an urgent need to develop more efficient and standardized human evaluation protocols.
Selection of papers: We consider existing summarization datasets with an average summary length of at least 150 words, which includes several popular datasets like arXiv [33], BillSum [127] and MultiNews [53]; see Table 4.1 for a full list. For our survey, we select all papers that evaluated summarization models using at least one of these datasets. All of these papers were published between June 2018 and September 2022, after the first long-form summarization datasets were released (PubMed / arXiv). Most of the 162 surveyed papers were published in major NLP/ML venues, but we also include newer preprints from 2022.

Long-form summaries are rarely evaluated by humans. We find that 101 out of 162 papers (62%) do not perform any human evaluation. 17 papers (11%) only perform human evaluation on short summaries (datasets like XSUM, [189]), for which human evaluation is much easier.

Human evaluation studies of long-form summaries are not reproducible. We further analyze the 44 papers performing human evaluation of long-form summaries to observe how often they follow reproducible practices from [64]. Overall, we find that most studies do not follow these guidelines. Only 2 of the 44 papers release their raw human annotation data for further analysis. Only 9 papers provide details of their annotator instructions or interface, and just 12 papers perform any kind of statistical analysis, despite most papers annotating less than 50 summaries. While 33 papers report using multiple annotators per summary, only 12 report inter-annotator agreement. Finally, just 14 papers conduct human evaluation on more than one dataset (more statistics in Appendix C.3).

Existing human evaluation setups lack standardization. In Table 4.2, we catalog the wide spectrum of human evaluation setups in the surveyed papers. 37 papers collect judgments of the full-length summary at once (“COARSE-grained”), while 6 papers collect judgments at a finer granularity such as sentences or entities (“FINE-grained”). Even within a granularity,

---

2We exclude five papers which used long-form summarization data for pre-training only, like [282].
setups differ: Likert-scale (24 papers), A/B testing (13 papers), binary per-sentence labels (4 papers) are the dominant protocols. In Section 4.3.1, we will see that this design decision is critical since COARSE annotations have much lower inter-annotator agreement than FINE.\(^3\)

**Human evaluation of long-form summaries is challenging and expensive.** Several of the surveyed papers discuss challenges in human evaluation of long-form summaries. 13 papers mention that expert annotators are necessary for human evaluation of long-form summaries, especially in technical domains like PubMed. 20 papers report that human evaluation of long-form summarization was *time-consuming, challenging, and expensive*, primarily due to the long length of the summary and source document. To tackle the issue of high annotator workload, we propose a partial annotation method in Section 4.3.2 and report high correlation to a full workload. Additionally, in Section 4.3.3 we investigate the usefulness of highlighting sentences to help annotators navigate the long source document. While this has been advocated for in short-form summary evaluation [82, 137] and used in 3 surveyed long-form papers, we find that it is only helpful when summaries are mostly correct and extractive.

### 4.3 The LONGEVAL guidelines for faithfulness human evaluation

In Section 4.2, we report several concerning issues with current human evaluation practices in long-form summarization. To move towards more efficient, reproducible and standardized protocols for human evaluation, we develop the LONGEVAL guidelines (Section 4.3.1-4.3.3, see Figure 4.1 for an overview). We focus on human evaluation of *faithfulness*, which [273] define as:

> “Checking the factual errors in the summary, where a factual error is a statement that contradicts the source document, or is not directly stated, heavily implied, or logically entailed by the source document”

---

\(^3\)Besides granularity, we also observe a large spectrum of annotator qualifications in our survey, ranging from MTurkers to expert graduates (Appendix C.3). Since non-experts are known to be unsuitable for this task [66, 55], we use experts in our work (Appendix C.2).
<table>
<thead>
<tr>
<th>Dataset</th>
<th>source (words)</th>
<th>summary (words)</th>
<th>papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>PubMed [33]</td>
<td>3092</td>
<td>205</td>
<td>59</td>
</tr>
<tr>
<td>arXiv [33]</td>
<td>5906</td>
<td>163</td>
<td>55</td>
</tr>
<tr>
<td>BillSum [127]</td>
<td>1284</td>
<td>174</td>
<td>19</td>
</tr>
<tr>
<td>MultiNews [53]</td>
<td>2103</td>
<td>263</td>
<td>54</td>
</tr>
<tr>
<td>GovReport [97]</td>
<td>7551</td>
<td>547</td>
<td>16</td>
</tr>
<tr>
<td>BookSum [138]</td>
<td>5102</td>
<td>505</td>
<td>4</td>
</tr>
<tr>
<td>SummScreen [28]</td>
<td>6965</td>
<td>227</td>
<td>11</td>
</tr>
<tr>
<td>SQuALITY [273]</td>
<td>5194</td>
<td>227</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.1: List of long-form summarization datasets considered in our survey along with average source document and summary lengths. Each dataset considered has at least 150 word summaries on average.

We conduct human annotation studies to empirically motivate LONGEVAL. Our experiments are on two long-form summarization datasets spanning diverse domains and levels of abstractiveness:

(1) **SQuALITY** [273] is a summarization dataset in the literary domain (avg. summary length of 227 words) where summaries describe the plots of English science fiction stories. SQuALITY is highly abstractive: on average just 16% of bigrams in the summary are present in the source document. We closely follow the human evaluation setup in [273], and use BART [154] and BART-DPR [113] as our summarization models along with human-written summaries.

(2) **PubMed** [33] is a summarization dataset in the scientific domain (avg. summary length of 205 words) that pairs English biomedical articles from PubMed⁴ with their abstracts as summaries. Compared to SQuALITY, PubMed is more extractive: 54% of summary bigrams are present in the source. We use BigBird-PEGASUS-large [306] and LongT5-large [76]

<table>
<thead>
<tr>
<th>Type of human evaluation</th>
<th># papers</th>
<th>% papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>101</td>
<td>62%</td>
</tr>
<tr>
<td>Short-form summaries only</td>
<td>17</td>
<td>11%</td>
</tr>
<tr>
<td>Likert-scale COARSE-grained</td>
<td>24</td>
<td>15%</td>
</tr>
<tr>
<td>A/B testing COARSE-grained</td>
<td>13</td>
<td>8%</td>
</tr>
<tr>
<td>Extrinsic evaluation</td>
<td>1</td>
<td>1%</td>
</tr>
<tr>
<td>Binary per sentence FINE-grained</td>
<td>4</td>
<td>2%</td>
</tr>
<tr>
<td>QA-based FINE-grained</td>
<td>2</td>
<td>1%</td>
</tr>
</tbody>
</table>

Table 4.2: Human evaluation setup in 162 summarization papers that evaluate long-form summaries. 73% of the papers do not evaluate long-form summaries with humans, while others vary significantly in their setups.

as our summarization models,\(^5\) along with human written summaries. By default, LongT5 / BigBird were highly extractive compared to human-written PubMed summaries (87% / 74% vs 54% bigram overlap with source). Hence, for half the generations we block 6-grams from being copied from the source,\(^6\) reducing extractiveness to \(~54\%\). We call this setting “PubMed-ngram-block”.

4.3.1 RQ1: Does inter-annotator agreement improve using fine-grained annotations?

In Section 4.2, we found that the dominant paradigm in literature (37 out of 44 papers) is to evaluate the whole summary at once (“COARSE”-grained, Figure 4.1 top left). 6 papers instead obtain fine-grained annotations for individual units (e.g., sentences) and average them (“FINE”, Figure 4.1 top right). Intuitively, FINE annotation has many advantages for longer summaries — it is less subjective than COARSE, since shorter spans needs to be judged rather than a long summary, and it helps localize model errors. However, the distinction between COARSE and FINE is never justified in literature, and inter-annotator agreement is rarely reported to understand the task subjectivity in each setup. To better

---

\(^5\)LongT5 is the best publicly available PubMed summarizer. BigBird is a popular long-form summarization baseline.

\(^6\)Reducing extractiveness / copying is also a suggestion for fair-use of copyrighted work [83, 260].
understand the tradeoff, in this section we conduct human evaluations annotating the same set of summaries using these two different protocols.

**Task formulation:** Let $F_{\text{summ}}$ denote the faithfulness score of a summary. For COARSE, $k$-point Likert scale ratings are obtained for the summary ($F_{\text{summ}} \in \{0,1,...k\}$), based on the faithfulness definition provided earlier. For FINE, we collect binary judgments of individual units in the summary and average them,
\[ F_{\text{sum}} = \frac{1}{|C_{\text{sum}}|} \sum_{c \in C_{\text{sum}}} F_c, \quad F_c \in \{0, 1\} \]

where \( C_{\text{sum}} \) is a set of units in the summary and \( F_c \) is the faithfulness judgment for the unit \( c \). In both protocols, the faithfulness score of a system is defined as 
\[ \frac{1}{|S|} \sum_{\text{summ} \in S} F_{\text{summ}} \]
where \( S \) is the set of summaries generated by the system.\(^7\)

While sentences are a popular granularity for FINE (4 of the 6 surveyed papers), we found that summary sentences in both datasets were overloaded with information. Hence, we segment sentences on conjunctions and punctuation to obtain more atomic units as \( C_{\text{sum}} \). These units are often clauses,\(^8\) similar to summary content units (SCUs) in Pyramid [191].

**Collecting COARSE annotations:** For SQuALITY, we re-use the annotations provided by [273] for faithfulness assessments. In their data, three annotators give each summary a 1-100 direct assessment rating [21]. Annotators with experience in professional copyrighting and editing were hired on Upwork,\(^9\) and these annotators were also involved in the creation of SQuALITY. Unfortunately, none of the surveyed papers that reported human evaluation results on PubMed released their raw human annotations.\(^10\) Hence, we collect our own COARSE evaluations on PubMed summaries on Upwork, using freelancers with professional experience reading and writing research papers (details in Appendix C.2.2). We collect 3 annotations per summary and use a 5-point Likert scale, the most common choice for COARSE assessment in our survey (18 out of 38 papers). In total, 120 summaries are evaluated.

\(^7\)We assume all summary units get an equal weight. However, some units may be more important than others, we discuss this in the Limitations section.

\(^8\)An even finer granularity is entities / numbers. We avoid this due to prohibitive annotation cost on long summaries.

\(^9\)https://www.upwork.com/

\(^10\)In our email correspondence with authors of these works, they mentioned losing access or compliance issues as reasons for not sharing human evaluations. We received some examples from [78] and [109] for reference.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>COARSE</th>
<th>FINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQuALITY</td>
<td>18.5</td>
<td>6.8</td>
</tr>
<tr>
<td>PubMed</td>
<td>11.8</td>
<td>7.3</td>
</tr>
<tr>
<td>PubMed + ngram block</td>
<td>11.7</td>
<td>9.3</td>
</tr>
<tr>
<td>Average</td>
<td>14.0</td>
<td>7.8</td>
</tr>
</tbody>
</table>

Table 4.3: Average standard deviation of faithfulness scores across annotators on a 100-point rating scale. Lower variation means higher agreement. Overall, we find that FINE-grained annotations have higher inter-annotator agreement than COARSE-grained annotations. Note that all FINE units of a summary were annotated to obtain these results ($f = 1.0$ in Section 4.3.2).

**Collecting FINE annotations:** For both SQuALITY and PubMed, we collect FINE annotations on Upwork (3 annotators per FINE unit) for the same set of 120 summaries evaluated using COARSE annotations. For SQuALITY, we hire freelancers with professional experience in English, creative writing, or education. For PubMed, we hire freelancers with prior experience analyzing biomedical articles. See Appendix C.2.1 for details of our annotator screening process, compensation, instructions, and screenshots of our annotation interface.

**FINE annotations have higher inter-annotator agreement than COARSE annotations.**

**This leads to more confident downstream estimates.** We present our results in Table 4.3. Overall, we observe that across all settings, FINE annotations have lower standard deviation (and thus higher agreement) in faithfulness scores than COARSE annotations (7.8 vs 14.0 average on 100-point scaled ratings). To illustrate the importance of higher agreement, we measure its effect on two downstream statistics that human evaluation is primarily used for: (1) correlation with automatic metrics; and (2) mean system performance. We adapt the bootstrap resampling analysis\(^\text{11}\) of [39] to estimate confidence intervals of these two downstream statistics for COARSE and FINE.

\(^{11}\)We slightly modify the algorithm in [39] for inter-annotator variance, see Appendix C.1.
Figure 4.4: Accuracy and variance after annotating a fraction of units per summary (X-axis) with FINE. Despite annotating just a fraction of the summary, we observe a high segment-level Kendall tau correlation with a full annotation (left). However we observe higher inter-annotator variance as the fraction reduces (right). Confidence intervals shown are 95% and computed across 1000 random subsets (see Appendix C.6 for left plot with Pearson).

In Figure 4.2, we plot the 95% confidence intervals of the Pearson correlation of various automatic evaluation metrics against FINE-grained and COARSE-grained human evaluation data. Across both datasets, FINE data leads to much narrower confidence intervals (0.15 vs 0.35 average uncertainty in Pearson correlation on PubMed) for the same number of summaries, implying higher statistical power. In Figure 4.3, we observe a similar trend with mean system performance. Interestingly, both annotation methods give the same relative ordering of systems (human > bart-dpr > bart for SQuALITY, human > longT5 > BigBird for PubMed-block), confirming the alignment of FINE and COARSE judgments on average.

**Recommendation:** Unlike the dominant trend in prior work, FINE-grained evaluations should be preferred over COARSE grained evaluation for long-form summaries. FINE annotations have lower inter-annotator variance than COARSE annotations and help localize model errors. In our setup we assume all FINE units are equally weighted while aggregating them to the final summary score. Despite this assumption, in our results we observe a consistent relative ordering of systems/metrics between COARSE and FINE annotations. Nevertheless, non-
uniform weighing of units is an interesting future work direction; more in the Limitations section.

4.3.2 RQ2: Can we reduce annotator workload by partially annotating a long summary?

In Section 4.3.1, we found that FINE annotations have lower variance than COARSE annotations. However, long summaries may be composed of several units (sentences or phrases) which each require FINE annotation. This could make FINE annotation very expensive for longer summaries (as also noted in our survey). What if we instead annotate a random subset of units from the summary? While this will lower annotation cost, how accurate would these partial annotations be? We explore this tradeoff by re-using the annotations collected in Section 4.3.1. For every summary, we randomly sample a fraction of units $f \in \{0.1, 0.2\ldots 0.9\}$ and then measure its correlation to the full set of annotations collected. Each annotator gets a different random sample of units for the same summary. In initial experiments, we found that this yielded higher accuracy than when keeping the same set of units per annotator.

Partial annotation has a high correlation to full annotation, but higher variance: In Figure 4.4 (left) we plot the segment level Kendall’s $\tau$ correlation (relative ordering of summary scores) between a partial annotation and full annotation for different values of $f$. Overall, we observe a high correlation across different values of $f$. Despite annotating just half the summary ($f = 0.5$), in both datasets we observe a high correlation of $0.78$-$0.89$ Kendall’s $\tau$ (95% interval) with a full annotation. Does a partial annotation preserve the variance benefits of FINE vs COARSE? In Figure 4.4 (right) we plot the inter-annotator variance for different values of $f$. In both datasets we find that a partial annotation has a higher variance than a full annotation. While for all values of $f$ in SQuALITY we find that FINE annotations still have lower variance than COARSE, in PubMed COARSE has lower variance than FINE for $f \leq 0.3$ with 95% confidence.
**Recommendation**: Having annotators judge a random subset of units in a long-form summary is a simple way to reduce FINE annotation cost, and has high correlation with a full annotation. However, it increases inter-annotator variance. Annotating 50% of the summary results in 0.78-0.89 Kendall’s $\tau$ correlation, with a 30-40% increase in standard deviation compared to full FINE annotation. Partial annotation may be limited in its ability to identify issues in summaries with very few errors. However, we find that this is not the case in current systems, which are abundant in faithfulness errors.

### 4.3.3 RQ3: Is it useful to align summary units to sentences in the source document?

So far, we have focused on design decisions on the summary side of evaluation. However, evaluating faithfulness requires a comparison of facts between a summary and a *source document*. Long-form summaries tend to have long source documents (Table 4.1): 3.1K words for SQuALITY and 5.1K words for PubMed. In Section 4.2, we found several mentioned human evaluation is challenging since annotators need to read long source documents. Some prior work has suggested highlighting spans in the source document that align with the summary [82, 137, 263] to assist human evaluators as shown in Figure 4.1. However, these efforts have exclusively focused on news summarization with relatively short source documents, like CNN/DM (804 words) [188] or XSUM (438 words) [189]. How useful is highlighting based on alignment, or “*hints*”, when the spans are chosen from much longer documents?

**What is the best highlighting algorithm?**

We conduct a study to identify the alignment algorithm best suited for highlighting hints. We manually annotate 125 FINE units from human-written summaries of the SQuALITY validation split, marking the sentences best supporting them from the source document. We then test several candidate methods for linking summary units to the source document. These include token overlap methods like ROUGE [161], retrievers [113], and fact verifiers [269]. In Table 4.4, we find that SuperPAL [52], a weakly supervised linking algorithm, performs
Table 4.4: A comparison of algorithms finding the top source document sentences for summary units in SQuALITY. R@k (recall@k) denotes the fraction of times the gold sentence was in the top-k predictions.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>R@3</th>
<th>R@5</th>
<th>R@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>BM25 [220]</td>
<td>0.38</td>
<td>0.46</td>
<td>0.56</td>
</tr>
<tr>
<td>ROUGE-1 [161]</td>
<td>0.31</td>
<td>0.34</td>
<td>0.46</td>
</tr>
<tr>
<td>ROUGE-L [161]</td>
<td>0.30</td>
<td>0.32</td>
<td>0.42</td>
</tr>
<tr>
<td>SIM [287]</td>
<td>0.37</td>
<td>0.52</td>
<td>0.60</td>
</tr>
<tr>
<td>DPR [113]</td>
<td>0.29</td>
<td>0.31</td>
<td>0.41</td>
</tr>
<tr>
<td>BERTScore-DB-XL [313]</td>
<td>0.30</td>
<td>0.37</td>
<td>0.46</td>
</tr>
<tr>
<td>SummaC-NLI [144]</td>
<td>0.22</td>
<td>0.26</td>
<td>0.34</td>
</tr>
<tr>
<td>MultiVers-FEVER [269]</td>
<td>0.47</td>
<td>0.58</td>
<td>0.71</td>
</tr>
<tr>
<td>SuperPAL [52]</td>
<td>0.61</td>
<td>0.68</td>
<td>0.77</td>
</tr>
</tbody>
</table>

Table 4.5: Annotator performance (accuracy, agreement, median time) in detecting summary errors with different types of source document highlight hints. Overall, we see little difference across the three settings.

<table>
<thead>
<tr>
<th>Hints</th>
<th>Acc. (↑)</th>
<th>Agree. (↑)</th>
<th>Time (secs) (↓)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(2-way)</td>
<td>(Fleiss)</td>
<td>All</td>
</tr>
<tr>
<td>None</td>
<td>93%</td>
<td>0.71</td>
<td>41.4</td>
</tr>
<tr>
<td>SuperPAL</td>
<td>92%</td>
<td>0.64</td>
<td>48.2</td>
</tr>
<tr>
<td>Gold</td>
<td>92%</td>
<td>0.63</td>
<td><strong>40.4</strong></td>
</tr>
</tbody>
</table>

To improve precision, we filter matches scoring less than 0.3 on SuperPAL, and show at most five highlights.

**Do highlighted hints improve summary error detection?** To answer this question, we manually perturb 50 fine summary units in SQuALITY validation summaries, introducing entity errors or negations like [137]. We modify the summary context of the perturbed unit to ensure summaries are self-consistent. Annotators are shown 50 perturbed and 50 un-perturbed summaries, and asked to annotate whether the summary units are faithful to the source in three settings: 12 (1) no highlighted hints; (2) SuperPAL highlighted hints; (3) gold

---

12To prevent any bias, each annotator receives only one of these settings for a particular summary.
Q: Did you find the highlighted hints useful while making your judgment?

TL;DR: 4 out of 5 annotators said Sometimes, 1 said Yes. More useful for SQuALITY, summary units copied verbatim from source, correct summaries.

“With summaries that had poor correctness, the hints were often a mess, and even correct spans had to be carefully checked. In summaries that were more correct, I could often just read the span and remember that it was correct, and then the hints helped me find the right source position, or refresh my memory about details.”

“They were more useful when the summary was a near verbatim source reproduction.”

“Yes, they were useful. Often they would highlight the exact passage needed to support the summary span.”

“In PubMed, they were a little more chaotic, even for good summaries.”

“SQuALITY summaries consisted of sentences or parts of sentences taken straight from the story (wording was exactly as in the text). So hints often lead to the exact place.”

“For SQuALITY, they were mostly accurate and helpful. For PubMed, they were less accurate and relevant.”

Q: Would the highlights have been sufficient to make judgments, or was reading the entire source document necessary?

TL;DR: 3 out of 5 annotators said No, 2 said sometimes in SQuALITY. Reading the entire document was critical.

“Reading the entire source document was very helpful to understand the basic story plot.”

“Even when the hints were relevant, sometimes they left out information (like character name)...”

“Initially I tried skimming ... then concluded it’s easier to read the entire document first.”

“With SQuALITY there were cases where almost all of the highlights did not make any sense and nothing of that was even mentioned in the story. With PubMed, it was even more difficult to find hints that support the text”

“No, much more useful to read the entire document along with keyword searches.”

Q: Did you use Ctrl+F searches in the source document while making judgments?

TL;DR: 4 out of 5 annotators said Yes, 1 said yes only for PubMed. Ctrl+F helped locate synonyms, entities.

“Yes, all the time. It was usually a safer bet than using the hints. The hints are given out of context of the whole SQuALITY story. There were a lot of problems with the PubMed hints involving numbers, which I often searched for. They were very rarely supported by the document, or contained wrong symbols (= instead of >).”

“Yes, mostly in cases the highlight did not support the summary unit partially or entirely,”

“I used Ctrl+F when looking for very specific words, like names. Searching was less helpful when it came to words that had synonyms or emotions.”

“I did Ctrl+F on keywords taken directly from the summary unit as well as synonyms and any specific words that I remembered from the story that could help me get to that place in the source document quickly.”

Table 4.6: Results and snippets from our questionnaire with FINE annotators. Overall, annotators find hints only sometimes useful, and mention reading the entire source document along with keyword searches.

<table>
<thead>
<tr>
<th>Question &amp; TL;DR response</th>
<th>Response Snippets</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q: Did you find the highlighted hints useful while making your judgment?</td>
<td>“With summaries that had poor correctness, the hints were often a mess, and even correct spans had to be carefully checked. In summaries that were more correct, I could often just read the span and remember that it was correct, and then the hints helped me find the right source position, or refresh my memory about details.”</td>
</tr>
<tr>
<td>TL;DR: 4 out of 5 annotators said Sometimes, 1 said Yes. More useful for SQuALITY, summary units copied verbatim from source, correct summaries.</td>
<td>They were more useful when the summary was a near verbatim source reproduction.”</td>
</tr>
<tr>
<td>Q: Would the highlights have been sufficient to make judgments, or was reading the entire source document necessary?</td>
<td>“Reading the entire source document was very helpful to understand the basic story plot.”</td>
</tr>
<tr>
<td>TL;DR: 3 out of 5 annotators said No, 2 said sometimes in SQuALITY. Reading the entire document was critical.</td>
<td>“Even when the hints were relevant, sometimes they left out information (like character name)...”</td>
</tr>
<tr>
<td>Q: Did you use Ctrl+F searches in the source document while making judgments?</td>
<td>“Reading the entire source document was essential to understanding the whole process, the hints in isolation were not good enough. The hints and the summary often confused similar objects, especially when pronouns were involved, from different parts of the source. In PubMed a similar thing happened when the source discussed what other papers had done – punctuation, acronyms, and abbreviations played a big role in providing context.”</td>
</tr>
<tr>
<td>TL;DR: 4 out of 5 annotators said Yes, 1 said yes only for PubMed. Ctrl+F helped locate synonyms, entities.</td>
<td>“Reading the entire source document was very helpful to understand the basic story plot.”</td>
</tr>
</tbody>
</table>

hints manually annotated by us. In Table 4.5, we show accuracy, inter-annotator agreement, and median time\textsuperscript{13} for each setting.

\textsuperscript{13}Calculated using the method in [5].
Highlighted hints have almost no effect in evaluating long-form summaries: Surprisingly, we observe that in all three metrics (accuracy, agreement, median time taken), scores are quite similar across the three settings. In fact, the “no-hint” setting scores slightly higher than the SuperPAL hint settings (93% vs 92% accuracy, 0.71 vs 0.64 Fleiss $\kappa$) and takes annotators less time (41.4 vs 48.2 seconds per unit). However, we find that hints helped annotate the first few units of a summary quicker (84.6 secs vs 115.6 secs per unit). We attribute our findings to a learning effect over time. FINE annotation of long-form summaries requires annotation of several units for the same document - summary pair. As annotation progresses, annotators get more familiar with the contents of the source document and summary, reducing the need for hints over time. See Appendix C.5 for learning trajectory plots.

Questionnaire with FINE annotators confirm limited utility of hints: Our evaluation so far is limited to perturbed human summaries. How effective are hints on model-generated summaries? To answer this, we ask five of our FINE Upwork annotators (from Section 4.3.1) a set of three questions about their experiences using highlighted hints. Detailed questionnaire results along with answer snippets are shown in Table 4.6. Overall, annotators find hints were useful only sometimes. Hints were less useful when (1) the summary unit was not supported in the source; (2) the summary unit was highly abstractive compared to the source; (3) pronouns, numbers, or abbreviations were involved; and (4) Pubmed summaries were annotated. Almost all annotators said it was necessary to read the entire source document before annotation to get an overall idea of the plot and resolve coreferences. Nearly all annotators used “Ctrl+F” searches along with hints to search for specific keywords while making judgments. This was especially true when the summary unit was incorrect, since the source document had to be thoroughly searched (beyond the hints) before confidently marking “Incorrect”.

---

14The FINE annotations in Section 4.3.1 were shown hints in the source document. Since hints may not be helpful, annotators were told not to solely rely on hints for annotation.
**Recommendation:** In contrast to recommendations in prior work, automatically highlighted hints are useful only in some specific cases of long-form summarization: mostly correct summaries, almost verbatim copied sentences. Annotators should be instructed to read the entire source document and to not rely solely on highlighted hints, since that could bias their judgments. Based on a small-scale study, we found SuperPAL [52] to be the most accurate method for finding hints, but its performance (61% recall@3) is far from ideal.

4.3.4 To what extent do our findings generalize to short-form summarization?

In this work, we exclusively focus on summarization datasets with an average summary length of at least 150 words. This constraint excludes two popular benchmarks in summarization research over the last five years: CNN/DM [188] and XSUM [189]. How relevant are our research questions (RQs) and findings for these short-form summarization benchmarks?

On average, XSUM (24 words) and CNN/DM (60 words) contain much shorter summaries than SQuALITY (237 words). XSUM outputs typically contain only 1 sentence or roughly 2-3 fine units per summary. This blurs the distinction between fine and coarse units, which makes it less useful to study RQ1 in these short-form settings. The shorter length of outputs also implies that evaluation is less expensive and consumes less time, which makes our RQ2 less relevant. Finally, on average, XSUM (440 words) and CNN/DM (800 words) also have much shorter source documents than datasets like SQuALITY (5200 words), reducing the need for alignment (the main premise for RQ3). The main motivation behind our study is that human evaluation of long-form summarization datasets like SQuALITY and PubMed is challenging and expensive due to the long length of the generated text. Overall, our research questions and findings are more relevant for long-form summarization datasets than for short-form summarization datasets like XSUM and CNN/DM.
4.4 Related Work

A large body of recent work has focused on new automatic evaluation methods for summarization via NLI-based algorithms [56, 144] or QA-based algorithms [272, 54]. Our work focuses on the much less studied area of human evaluation, the gold standard for developing automatic metrics. A notable effort in this space is the Pyramid method [191], along with work improving Pyramid efficiency [230, 311]. Efficient Pyramid-like protocols have been used to collect large-scale datasets human judgments [18, 166] in short-form news summarization tasks like CNN/DM. While these efforts focus on salience evaluation and assume access to multiple references, our work focuses on faithfulness and operates in a reference-free setting. Moreover, we focus on long-form summarization tasks like SQuALITY and PubMed, which are much more challenging and expensive to evaluate.

Evaluating summary faithfulness relates to fact verification [264], where claim sentences are checked against a large knowledge source (Wikipedia). Prior work [187] attempts to simplify the human fact checking process by methods like knowledge source snippets [59], similar to hint highlights (§4.3.3). Faithfulness in summarization differs from fact verification in three ways: (1) summaries are paragraph-long and contextual compared to single sentence stand-alone claims in fact verification; (2) summaries are grounded to a source document, compared to a large knowledge source in fact verification; (3) summaries are model-generated compared to human-written claims in fact checking datasets [251, 268].

4.5 Conclusion

We present the LONGEVAL guidelines, a set of recommendations for moving towards standardized human evaluation of long-form summarization. We empirically analyze each recommendation on two datasets. Overall, we find that (1) fine-grained annotations have lower inter-annotator variance than coarse-grained annotations; (2) partially annotating a summary reduces annotator workload while maintaining accuracy; (3) highlighting hints in
the source document has limited usefulness for evaluating long-form summaries. As future work, we plan to conduct experiments on other aspects of summarization evaluation like salience and coherence.

4.6 Limitations

Human evaluation is a noisy process with many confounding variables. Some of these variables were kept constant among experiments on a dataset, but modifying them could change the trends in the results. These include: (1) number of annotations per summary; (2) the specific annotation interface used; (3) granularity for FINE evaluation (sentences vs phrases); (4) Number of points in the Likert scale for COARSE evaluation; (5) set of summarization systems evaluated; and finally (6) relative (eg: A/B tests) vs absolute evaluation (eg: Likert), which has been discussed in [246] for short-form news summarization datasets like CNN/DM.

Our paper is limited to faithfulness evaluation, but summaries are typically evaluated for salience, fluency, coherence as well [55]. While fluency may be less of an issue due to large-scale language model pretraining [46], coherence and salience are important aspects to evaluate especially in long-form summarization [71]. Our findings may not generalize to evaluation of coherence or salience.

Our experiments in Section 4.3.1 assigned an equal weight to each FINE unit while calculating the overall score of the summary. However, the faithfulness of some FINE units may be more important than others. A non-uniform weighing of FINE units may be a good strategy if there is a notion of how critical a particular unit is for a summary’s correctness. For example: (1) PICO units are critical in medical summaries [43]; (2) the Pyramid scheme [191] uses a reference frequency-based unit importance, assuming access to multiple gold references. However, a consistent notion of importance is difficult to
establish across different domains, and also depends on an individual consumer’s preferences. Designing non-uniform weighing schemes is an interesting direction for future research.

**Ethical Considerations**

All experiments involving human evaluation in this chapter were exempt under institutional IRB review. We fairly compensated each Upwork freelancer involved in this study, at a rate of 15-20$ per hour (respecting their suggested Upwork hourly wage). For each round of annotation, we estimated the average amount of time the task would take (by running pilots among ourselves), and provided annotators with the estimated time requirement. Most freelancers finished the task within the time window, but sometimes exceeded it by 0.5-1 hr. We compensated freelancers based on the actual time they took and their hourly wage, rather than a fixed amount per annotation.

**4.7 Impact / Retrospective**

This study was originally published in EACL 2023 [128], where it was recognized with an outstanding paper award.\(^{15}\) To encourage reproducible human evaluation setups, I also released **LONGEVAL** as an easy to use Python library, which has received over 3000 downloads in the last seven months.\(^{16}\) Among its eight follow-up studies (as of August 2023), two notable follow-ups are: (1) FineGrained-RLHF [296], a study empirically showing the benefits of fine-grained human evaluation in the sample efficiency of RLHF training; (2) FActScore [182], a new automatic metric for long-form text generation that uses large language models to annotate automatically generated fine-grained atomic facts to judge the output’s factual correctness.

---

\(^{15}\)12 papers out of 1600 submissions were awarded, see https://2023.eacl.org/program/best-paper.

\(^{16}\)https://pepy.tech/project/longeval
Careful human evaluation is becoming the bedrock of modern chatbots like ChatGPT through RLHF [225, 196]. I expect careful human evaluation to be an important research area in NLP in the next few years. I also anticipate several research and commercial efforts to collect large scale human feedback data (hopefully, fine-grained) on large language model outputs, similar to OpenAI’s private effort in the past few years. The primarily goal of this data would be train the next generation of aligned large language models.
Motivation and Summary: So far this thesis focused on improving correctness of model outputs, and developing better methods to evaluate it. As large language models continue to rapidly improve, there’s a growing fear of malicious use cases, such as fake content creation and academic plagiarism. This has motivated the development of approaches that identify AI-generated text, including those based on watermarking or outlier detection.

In this chapter I will introduce these techniques, as well as stress test detection algorithms to paraphrases of AI-generated text. To stress test these detectors, I build a 11B parameter paraphrase generation model (DIPPER) that can paraphrase paragraphs, condition on surrounding context, and control lexical diversity and content reordering. Using DIPPER to paraphrase text generated by three large language models (including GPT3.5-davinci-003) successfully evades several detectors, including watermarking, GPTZero, DetectGPT, and OpenAI’s text classifier. For example, DIPPER drops detection accuracy of DetectGPT from 70.3% to 4.6% (at a constant false positive rate of 1%), without appreciably modifying the input semantics.

To increase the robustness of AI-generated text detection to paraphrase attacks, I introduce a simple defense that relies on retrieving semantically-similar generations and must be maintained by a language model API provider. Given a candidate text, our algorithm searches a database of sequences previously generated by the API, looking for sequences that match the candidate text within a certain threshold. I empirically verify this defense using a database of 15M generations from a fine-tuned T5-XXL model and find that it can detect
In a shocking finding, a scientist discovered a herd of unicorns living in a remote valley. They have never been known to mingle with humans. Today, it is believed these unicorns live in an unspoilt environment, surrounded by mountains and protected by a thick clump of wattle trees, giving it a majestic appearance. Along with their so-called miracle of multicolored coat, their golden colored feathers make them look like mirages.

Some of them are rumored to be capable of speaking a large amount of different languages. They feed on elk and goats as they were selected from those animals that possess a ferociousness to them, and can "eat" them with their long horns.

There were never any reports of them mingling with people. It is believed they live in an unspoiled environment, surrounded by mountains and protected by a thick clump of wattle. The herd has a regal look to it, with the magic, rainbow-colored coat and golden feathers. Some of them are said to be capable of speaking many languages. They eat deer and goats, because they are the descendants of those animals that sprang from fierce, dangerous animals and have horns long enough to "eat" these animals.

80% to 97% of paraphrased generations across different settings while only classifying 1% of human-written sequences as AI-generated. I have open-sourced the pretrained models, code, and data associated with this chapter.¹

5.1 Introduction

Large language models (LLMs) such as ChatGPT [225] exhibit an unprecedented ability to write coherent and relevant long-form text in response to user-specified prompts. These abilities have sparked fears of malicious applications such as automatically generating fake news articles or homework answers [239]. To defend against these use cases, several algorithms have recently been proposed to detect AI-generated text, including watermarking [121], GPTZero [252], DetectGPT [184], and OpenAI’s text classifier [194]. However, it remains unclear how robust these algorithms are to paraphrase attacks, in which AI-generated text from an LLM is rewritten by another (smaller) model to convey approximately² the same meaning but using different word choice and syntax.

¹https://github.com/martiansideofthemoon/ai-detection-paraphrases
²We use the quasi-paraphrase definition of semantic equivalence [17] in this chapter.
In this chapter, we first demonstrate the vulnerability of these existing detectors to paraphrase attacks (Section 5.3, 5.4). Such attacks require an external paraphraser model, since paraphrases generated by the base LLM are still susceptible to detection techniques such as watermarking. We train an 11B parameter paraphrase generation model called DIPPER (or Discourse Paraphraser) to execute these attacks. DIPPER possesses two unique features that help its outputs evade AI-generated text detectors: (1) **Paraphrasing long-form text in context**: Most modern paraphrasers are exclusively trained on sentence-level data, ignoring discourse-level information. However, many critical use cases of LLMs involve generating long-form text as responses to detailed user-specified prompts. Thus, we train DIPPER to paraphrase paragraph-length texts, re-order content, and optionally leverage context such as user prompts; (2) **Controlling output diversity**: Another weakness of existing paraphrasers is that they lack an easy way to control output diversity. An attacker may want to apply just the minimum amount of paraphrasing to evade a detector. DIPPER provides users with two intuitive scalar control knobs at inference time (lexical diversity, content reordering) that are trained end-to-end.

We use DIPPER to attack several recently proposed AI-generated text detectors (see Figure 5.1 for an attack overview). Experiments on multiple tasks and LLMs (including GPT3.5-davinci-003) show that after paraphrasing with DIPPER, a substantial fraction of AI-generated texts are misclassified as human-written texts by all detectors. For example, DetectGPT [184] correctly detects 70.3% of AI-generated sequences from GPT2-XL, but after paraphrasing, its detection rate drops to only 4.6%\(^3\) despite minimal semantic modification. We confirm the validity of DIPPER’s paraphrases through several automatic evaluations and a human evaluation of semantic similarity.

Given the vulnerability of AI-generated text detectors to paraphrasing, how can we defend against such attacks? In the second part of this chapter (Section 5.5), we propose to

\(^3\)These detection rates were computed at a constant false positive rate (FPR) of 1%. Due to the importance of low FPR in this task, we recommend using a fixed low FPR rather than AUCROC values; see Section 5.4.1.
use retrieval methods to detect AI-generated text instead of relying on statistical properties of the text or watermarking. First, an LLM API provider stores every output generated by their model in a database. The API provider then offers a service in which a semantic representation of a candidate text is compared to representations of every generation stored in the database. The search focuses on the semantics of the input and can leverage both standard IR methods such as BM-25 [220] as well as semantic vector representations such as P-SP from [289]. Since paraphrasing does not modify the semantics of the input, this algorithm is robust to paraphrasing attacks. Specifically, we find that 97.3% of PG19 paraphrases and 80.4% of Wikipedia paraphrases are successfully detected in a large database of over 15M generations, at a 1.0% false positive rate.

In contrast to concurrent work that also uses paraphrasing to attack AI-generated text detectors [223], our work offers more comprehensive attack experiments, a new and more powerful paraphraser, human evaluations of paraphrase quality, and finally a novel defense mechanism based on retrieval to combat such attacks. To spur future research in this area, we will release our DIPPER model, data, and a codebase for evaluating both existing detectors and our retrieval-based method.

5.2 A background on detectors of AI-generated text

In this section, we provide an overview of existing algorithms that have been developed for the purpose of detecting machine-generated text. Such algorithms fall into three main categories: (1) watermarking algorithms, which modify the generative algorithm to encode hidden information unique to the API (Section 5.2.1); (2) statistical outlier detection methods, which do not modify the generative algorithm but look for inherent artifacts in generated text (Section 5.2.2); and (3) classifiers trained to discriminate machine-generated text from human-written text (Section 5.2.3). Finally, in Section 5.2.4, we compare and contrast our work to [223], who also note the efficacy of paraphrasing attacks but do not consider a
retrieval-based defense in their pessimistic conclusion about the fate of AI-generated text detection.

5.2.1 Watermarking language model outputs

A “watermark” is a modification to the generated text that can be detected by a statistical algorithm while remaining imperceptible to human readers. Effective watermarks are difficult to remove and have little effect on the quality of generated text. Prior work attempted to watermark natural language using syntax tree manipulations [257, 178], and this area has gotten renewed interest with large language models generating human-like text [3, 73]. Most recently, [121] propose a simple algorithm that only requires access to the LLM’s logits at each time step to add watermarks. The watermark can then be verified with only blackbox access to the LM and knowledge of a specific hash function. This algorithm operates in three steps:

1. **Mark a random subset of the vocabulary** as “green tokens” (or tokens representing the watermark, as shown in Figure 5.1) using the hash of the previously generated token as a random seed. A total of $\gamma|V|$ tokens are marked green where $\gamma$ is the fraction of the tokens that are watermarked with default $\gamma = 0.5$.

2. **Increase the logit value** for every green token by a constant $\delta$ (= 2 by default), which denotes the watermark strength. This raises the probability of sampling green watermarked tokens, especially for high-entropy distributions.

3. **Sample sequences** using decoding algorithms such as nucleus sampling [89], leveraging the modified probability distribution at each timestep before truncation.

**Detecting the watermark**: Verifying whether a text is generated by a watermarked LM is possible with just knowledge of the hash function and tokenizer. Specifically, the verifier tokenizes the text and counts the number of green tokens it contains. This is used to calculate the standard normal score ($z$-score) for the hypothesis test. If the sequence with $T$ tokens
contains a certain number of the green token (denoted as $|s_G|$), the $z$-score can be computed by:

$$z = \frac{(|s_G| - \gamma T)}{\sqrt{T\gamma(1-\gamma)}}$$

Intuitively, a higher $z$-score implies it is less likely for a human to have written the text (null hypothesis) since it contains a higher than expected number of green tokens. [121] recommend using a high $z$ value ($z > 4$, or $p < 3 \times 10^{-5}$) to reduce the risk of false positives (human-written text classified as AI-generated). Low false positive rates are critical in AI-generated text detection algorithms [194]—we discuss this in Section 5.4.1.

5.2.2 Statistical outlier detection methods

Unlike the watermarking algorithms, outlier detection algorithms make no modification to the generative algorithm. Instead, they attempt to distinguish between human-written and machine-generated text based on the presence of artifacts in generated text [226, 89]. Early methods detect statistical irregularities in measures such as entropy [147], perplexity [16], and $n$-gram frequencies [72, 10]. After the release of GPT-2, [65] introduced the GLTR visualization tool to assist human verifiers in detecting machine-generated text. Most recently, the release of ChatGPT has prompted the development of two new tools, namely a closed-source tool called GPTZero [252], and open-source DetectGPT [184]. DetectGPT uses an observation that model-generated text lies in the negative curvature regions of the model’s log probability function. It constructs multiple perturbations of the model generated text (using a mask-and-fill strategy), and compares the log probability of the perturbations with the unperturbed generation. Text is considered model generated if the log probability of the unperturbed text is significantly higher than the log probability of perturbations.
5.2.3 Classifiers

The third class of detection methods relies on classifiers that are fine-tuned to distinguish human-written text from machine-generated text. Early efforts in this vein use classifiers to detect fake reviews [94] and fake news [308]. Other related studies examine classification performance across domains [12] and decoding strategies [98]. Such studies inspired others to use their insights to improve generative performance [38, 129]. Most recently, OpenAI fine-tuned a GPT model to perform this discrimination task and released it as a web interface [194]. They fine-tuned this classifier using generations from 34 language models, with text sourced from Wikipedia, WebText [208], and their internal human demonstration data.

5.2.4 Comparison to Sadasivan et al. (2023)

In very recent concurrent work, [223] also demonstrate the utility of paraphrasing attacks against AI-generated text detectors. While their work makes use of off-the-shelf sentence-level paraphrase models, DIPPER possesses advanced discourse-level rewriting capabilities as well as fine-grained diversity control, which allows us to thoroughly analyze the effectiveness of various paraphrasing strategies. Our experiments also encompass more tasks, datasets, and detection algorithms. Moreover, we evaluate larger language models like GPT3.5-davinci-003. Finally and most importantly, our retrieval-based defense directly contradicts the “impossibility result” of [223] and its associated proof, which states that even an optimal detector will approach the performance of a random classifier as the distance between the distributions of LLM-generated text and human generated text goes to zero. Since our detector does not rely on properties of the text but rather a corpus search, the quality of the generated text is irrelevant to the effectiveness of our detector, and thus their proof does not apply to our method.
to the two nuns when they came. Touching sight to my companion and stayed, but apparently I was a very position or how long I might have palpitated violently and in a moment I received more consolation; my heart

p3

My companion prayed kneeling, but I bowed myself down, my forehead touching the bottom step of the altar and my arms stretched out on the steps above.

p4

I don't know how long I stayed in that position or how long I might have

My soul was in some measure relieved.

My companion prayed kneeling in an upright position, my forehead pressed on the bottom step of the altar; my arms stretched out on the steps above. I do not think I have ever addressed God with more fervour and received more consolation; my heart was pounding violently, and in an instant I was oblivious to everything around me. I do not know how long I stayed in that position or how long I might have stayed, but apparently I saw a very touching sight to my companion and to the two nuns when they came.

Figure 5.2: The method used to train DIPPER on English translations of the French novel The Nun. We first align sentences between the two translations to create parallel data. Next, a subset of the alignments are chosen; in this example, we use (p2, q2) and (p3, q3q4). We shuffle sentences, compute control codes, and fine-tune a T5-XXL LM to generate p2p3 given q3q4q2 and the context p1 and p4.

5.3 Building a controllable discourse paraphraser

Having outlined existing methods to detect AI-generated text, we now focus on a simple attack against all detection techniques: paraphrasing the generated text. Intuitively, paraphrasing alters the statistical properties of AI-generated text, which can fool outlier detection or classifiers while also reducing the number of watermarked tokens (Figure 5.1). To evade such detectors, a paraphraser must be able to handle context in the form of prompts or multi-sentence inputs. Its behavior should also be controllable in order to make as many/few changes as needed to evade a given detector. In all cases, it should not appreciably change the input semantics. Finally, to evade watermarking, the paraphraser must be different from the watermarked model, as otherwise the paraphrases will also be watermarked. Below, we detail how we construct a paraphraser (DIPPER) with all these properties.4

Constructing paraphrase data: Our process involves fine-tuning a LLM on a parallel dataset of paragraph-level paraphrases, which we modify to model control, external context

4To better ground DIPPER’s abilities in prior work, we survey existing paraphrase models in Appendix D.2.1.
and content reordering. We leverage the PAR3 dataset [248], which contains multiple translations of non-English novels into English aligned at a paragraph level, which we treat as paraphrases. More formally, let \( p \) and \( q \) be aligned paragraphs, where \( p_1, p_2, ..., p_N \) denote sentences of \( p \) and \( q_1, q_2, ..., q_M \) denote sentences of \( q \). Note that \( M \) may not be equal \( N \) when two translators disagree on when to merge and split sentences. We perform the following steps (overview in Figure 5.2):

1. **Align sentences** of \( p \) to sentences of \( q \) by using the semantic similarity scores from the paraphrase similarity metric in [287] to run the sequence alignment algorithm designed by [190] which uses dynamic programming (metric details in Section 5.4.1).

2. **Choose a subset of sentences** \( p_i, ..., p_j \) from the first paragraph. Let \( q_{i'}, ..., q_{j'} \) be the corresponding alignment in the second paragraph. In Figure 5.2, \( i = 2, j = 3, i' = 2, j' = 4 \).

3. **Re-order** the sentences \( q_{i'}, ..., q_{j'} \) randomly, and compute the **diversity control codes** between \( p_i, ..., p_j \) and shuffle\((q_{i'}, ..., q_{j'})\). We shuffle the sentences to allow for the model to learn content re-ordering. We compute lexical diversity (\( L \)) using unigram token overlap, and the order diversity (\( O \)) using the Kendall-Tau correlation of tokens of overlapping words between \( p_i, ..., p_j \) and shuffle\((q_{i'}, ..., q_{j'})\), also used in [134]. These scores are normalized to values \( \{0, 20, 40, 60, 80, 100\} \), where \( L = 20 \) roughly corresponds to a 20% lexical modification.

4. **Map** the shuffled \( q_{i'}, ..., q_{j'} \) to \( p_i, ..., p_j \), leveraging context from the rest of \( p \) and control codes using string concatenation. Let input = shuffle\((q_{i'}, ..., q_{j'})\). We map,

\[
\text{lexical} = L, \text{order} = O \oplus p_1, ..., p_{i-1} \oplus <p> \text{ input } <p> \oplus p_{j+1}, ..., p_N \rightarrow p_i, ..., p_j
\]

where \( \oplus \) is string concatenation. During inference, we can paraphrase any sequence of sentences by marking it with \(<p>\) tags, assigning the control codes \((L, O)\) the desired diversity values.
We fine-tune a sequence-to-sequence Transformer [262] on this data, initialized with the pretrained 11B parameter T5-XXL checkpoint [211]. See Appendix D.3.1 for details.

5.4 Experiments attacking detection algorithms with DIPPER

In this section, we describe our experimental setup in Section 5.4.1-5.4.2 and present our results in Section 5.4.3. Overall, we find that DIPPER evades all detectors across three LLMs (including GPT3.5).

5.4.1 Evaluation metrics

Detection accuracy: Our first metric measures how often the input text is correctly detected as AI-generated. Since detection rates are heavily dependent on the chosen detection threshold, the AUC-ROC metric is commonly used to measure detector performance [184], which considers the range of all possible thresholds. However, in this application, it is critical that the false positive rate (FPR) is low; in other words, human-written text must almost never be classified as machine-generated [194, 121]. Hence, we fix the FPR to 1% for all detection algorithms (although even 1% is likely too high in practice), and adjust the detection threshold accordingly while reporting detection accuracies. Additionally, we also plot ROC curves focusing on the 0-1% FPR region. Overall, we expect detection rates to plummet on paraphrased text at a constant FPR of 1%.

Semantic similarity (Sim): Detection accuracy is an insufficient evaluation of our attack’s success. We also need to measure whether the original and paraphrased generations share approximately the same semantics. We measure semantic similarity using the state-of-the-art semantic similarity model P-SP from [289], an embedding averaging model trained on a large corpus of filtered paraphrase data [288]. We consider semantics to be approximately preserved if the P-SP score is > 0.76, which is the median P-SP score of human-written paraphrase pairs on the PAR3 dataset [248]. Besides semantic similarity, we
Table 5.1: Performance of detection algorithms (at 1% FPR) before and after DIPPER paraphrasing on open-ended generation using Wikipedia prompts (300 generated tokens). As the diversity (L,O) increases, detection rates decrease across algorithms, with nearly perfect semantic similarity (Sim). *GPT3.5 DetectGPT scores computed using 200 samples at 20% FPR as it scores 0% at a 1% FPR.

conduct several automatic evaluations, ablation studies, and human evaluations of intrinsic paraphrase quality in Appendix 5.6.

5.4.2 Models, datasets & detection algorithms

**Base language models**: We conduct attacks on three language models of varying sizes that belong to different model families. We consider the GPT2-XL model (1.5B parameters) [208], the OPT-13B model [312], and the text-davinci-003 variant from the GPT-3.5 family [23], which has 175B parameters and has additionally been instruction tuned using reinforcement learning from human feedback [196]. For all LMs, we sample
generations that are 300 tokens long before passing them through DIPPER for the attack experiments.\textsuperscript{5}

**Natural language generation tasks:** We experiment with two long-form text generation tasks, since most malicious applications (e.g., fake article creation) are associated with long-form outputs. First, we consider *open-ended generation*, where an LM generates a continuation to a two-sentence prompt. To obtain our prompts, we sample 3K contiguous two-sentence chunks from the validation split of WikiText-103 \[179\] and use the next 300 tokens as the “human-written” continuation. Second, we evaluate *long-form question answering* \[57\], in which an LM answers a question with a 300-word answer (dataset details in Appendix D.3.2). The human reference answers or continuations are used only to adjust detection thresholds of studied methods to maintain a 1\% FPR.\textsuperscript{6}

**Detection algorithms:** We attack five detectors: (1) watermarking \[121\]; (2) Detect-GPT \[184\]; (3) GPTZero \[252\]; (4) OpenAI’s text classifier \[194\]; and (5) RankGen-XL-all \[129\].\textsuperscript{7} We use the default hyperparameters for each detector. We also respect their minimum length specifications, discarding instances where any of the AI-generated text, human-written text, or paraphrased text is shorter than the minimum length.

**Paraphrasing AI-generated text:** We pass the prompts for each task and AI-generated responses to those prompts through DIPPER. Specifically, we feed the model input of the form,

\[
\text{lexical} = L, \text{order} = O \text{ prompt } \langle p \rangle \text{ generated-text } \langle /p \rangle,
\]

where \( L \) and \( O \) represent the paraphraser diversity control codes and the \( \langle p \rangle \) and \( \langle /p \rangle \) special tokens mark the boundaries of the text to be paraphrased. We use \( L = 20, 40, 60 \)

\textsuperscript{5}For GPT2-XL and OPT-13B, we generate text using nucleus sampling \[89\] with \( p = 0.9 \). For davinci-003, we use the default hyperparameters on the API Playground (temperature = 0.7).

\textsuperscript{6}Evaluating how well the studied LMs perform on these two tasks is a challenging problem in its own right that could make additional use of the human references, but this is irrelevant to the chapter.

\textsuperscript{7}While RankGen was not explicitly optimized for this task, it was trained to identify well-written continuations, so we hypothesize that it could also act as a reasonable AI-generated text detector.
Table 5.2: Detector performance (at 1% FPR) on long-form QA before/after paraphrasing. As diversity (L,O) increases, detection rates decrease with very high semantic preservation (Sim). *GPT3.5 DetectGPT uses 100 samples at 20% FPR since it gets 0% performance at 1% FPR.

and $O = 0,60$ in our main attack experiments. After paraphrasing, we ensure that the AI-generated sequence, paraphrased sequence, and human-written sequence have an equal number of words by truncating them to the length of the shortest among the three. To ensure higher semantic preservation, we iteratively paraphrase long sequences three sentences at a time, keeping already paraphrased text in the context of the generation. For simplicity, we only paraphrase each generation once.\(^8\)

5.4.3 Attacking AI-generated text detectors

Our results in Table 5.1 and Figure 5.2 show that:

---

\(^8\)In Section 5.4.4, we discuss a sophisticated attack which samples multiple times, keeping the best paraphrase.
Paraphrasing significantly lowers detection accuracy while preserving input semantics. Across all LMs, detectors, and tasks, paraphrasing significantly lowers detection accuracy across all diversity control codes. For instance, paraphrasing GPT2-XL open-ended generations reduces watermark detection accuracy from 100% to 57.2%, and DetectGPT accuracy from 70.3% to just 4.6%. Trends are similar even for large LMs like GPT-3.5, for which paraphrasing reduces OpenAI’s text classifier accuracy from 30.0% to 15.6%. Additionally, DIPPER preserves semantics effectively, as 88%-99% paraphrases achieve a P-SP score higher than the median score of human-written paraphrases. Overall, we find that watermarking is the most resilient detector to paraphrasing.

Non-watermarking detectors are generally ineffective. We observe that all detectors apart from watermarking struggle with text generated by larger models like OPT-13B and GPT-3.5, achieving detection accuracies < 50%. While DetectGPT is effective on the smaller GPT2-XL model (74.9% on long-form QA), its accuracy drops to just 29.8% on OPT-13B. Furthermore, GPTZero and RankGen perform the worst among the five detectors on all tested LMs (Table 5.1), as they are only able to detect < 15% of non-paraphrased AI-generated text. Thus, we recommend against using these detectors.

ROC plots confirm the trends at different false positive rates. In Figure 5.5, we plot the detection accuracy (true positive rate) at different values of FPR between 0% and 1% for GPT2-XL. Overall, paraphrasing significantly drops detection rates across all FPR thresholds (more plots in Appendix D.5).

5.4.4 Alternative paraphrasing attacks

Here, we discuss two other (untested) ways to attack AI-generated text detectors via paraphrasing, which further showcase the brittleness of existing detectors.

---

9Except RankGen, which scores paraphrases as AI-generated more often than non-paraphrased text. We attribute this to paraphrases being poorer continuations to the prompt compared to the original (Appendix 5.6), an aspect RankGen bases its score on. However, it has low overall performance since it is not trained for this task.
Paraphrasing multiple times: Our presented attacks use just a single paraphrase generated by DIPPER to evade detection. A simple way to further improve the effectiveness of a paraphrase attack is to sample multiple times\(^\text{10}\) from DIPPER and choose a paraphrase that evades the detector while also preserving semantics. We do not perform this attack as it can only be done if an attacker has access to a detector, which may be a strong assumption (see Section 5.5.5). That being said, using multiple paraphrase samples can make the attacks even more potent against publicly available detectors.

Non-DIPPER paraphrasers: A second alternative is to use non-DIPPER paraphrasers that operate at the sentence level. These models can be deployed for long-form text inputs by paraphrasing the inputs sentence by sentence, ignoring prompt context. While the concurrent work of \cite{223} shows that this method can also evade detection, our ablations in Section 5.6 show that these paraphrasers have lower quality and are less compatible with the prompt as DIPPER paraphrasers. A more interesting option is to use a LLM served by an API to perform few-shot contextual paraphrasing. While this method is likely to provide accurate paraphrases,\(^\text{11}\) they may be detectable by strategies like watermarking (whether using the same API as the original LLM or a different one). We thus expect a sophisticated adversary to use their own private paraphraser (like DIPPER) to evade detection.

5.5 Defense against paraphrase attacks using retrieval

In Section 5.4.3, we showed that paraphrasing is an effective attack against AI-generated text detectors. How can LLM API providers defend against these attacks? In this section, we propose retrieval over previously-generated sequences as a defense against paraphrase attacks. At a high level (Figure 5.3), an API provider first stores every sequence generated

\(^{10}\)Precisely, compute \(f_{\text{dipper}}(x)\) for different random seeds while sampling text. Alternatively, an attacker could also compute \(f_{\text{dipper}}(f_{\text{dipper}}(...f_{\text{dipper}}(x)))\), but this will lead to excessive semantic drift from \(x\).

\(^{11}\)In initial experiments, we observed that DIPPER performs competitively with the much larger and more powerful GPT-3.5 davinci-003 model in terms of paraphrase quality, and significantly better at controlling diversity. This finding shows that specialized smaller models can outperform LLMs in paraphrasing tasks.
by their LLM in a database. The API provider offers an interface that allows users to enter candidate AI-generated text as a query. The interface searches over the entire database of previously-generated text, trying to find a sequence that approximately matches the content of the input query. This search can be done using a semantic similarity scorer like P-SP [289] or a retriever like BM25 [220]. Since paraphrasing approximately preserves input semantics, we expect such a defense to still be able to map paraphrased generations to their source. We formalize our detector in Section 5.5.1, and then conduct a controlled comparison with competing detectors in Section 5.5.2. We evaluate retrieval-based detection at scale using a large retrieval corpus of 15M generations in Section 5.5.3. In Section 5.5.4, we share ideas for enabling further scaling of retrieval-based detection. Finally, in Section 5.5.5 we extensively discuss limitations of retrieval-based detection.

5.5.1 Formulating the retrieval defense

Let $f_{LM}$ be an LLM API (e.g., GPT-3.5) that takes a prompt $x$ as input and returns a continuation $y$. Let $f_{ret}$ be an encoder (e.g., TF-IDF, neural network) that embeds variable-length sequences into fixed-size vectors that represent the input semantics. Then, we do the following:

**Building the database:** Let $x_1, ..., x_N$ be the set of prompts that have been fed as input to the API in the past with $y_i = f_{LM}(x_i)$ being the LLM output. Here $N$ can potentially be very large for popular APIs (we study up to $N = 15M$). We construct our database $Y = [y_1, ... y_N]$ by encoding every LLM API output with our retrieval encoder, or $y_i = f_{ret}(y_i)$. The database $Y$ is dynamically updated and stored on the API side. It is inaccessible to clients except via the API described in the next step.

**Querying the database:** Let $y'$ be a candidate text and $y' = f_{ret}(y')$ its encoded vector. Suppose a client wishes to know whether $y'$ was generated by the API $f_{LM}$. The API provider can check this by seeing whether the maximum similarity score of $y'$ to an entry in the database exceeds some detection threshold $L$ chosen by the API provider:
Figure 5.3: An illustration of AI-generated text detection with retrieval. Several users (including the attacker, shown as the purple emoji) feed prompts to the API which are collectively added to a private API-side database. Candidate queries are compared against this database using a retriever like BM25.

\[
\text{output} = \text{score} > L, \quad \text{where} \quad \text{score} = \max_{i \in \{1, \ldots, N\}} \frac{y' \cdot y_i}{|y'| \cdot |y_i|}
\]

We expect unperturbed machine-generated text to always get a score of 1.0, while paraphrasing the text may lower the detection score. Hence, lowering \( L \) will increase the detection rate of heavily-paraphrased text but also increase the false positive rate (i.e., human-written text that resembles sequences previously generated by the LLM API can be falsely flagged). Since \( N \) can be very large, the score can also be approximated using efficient nearest neighbor libraries like FAISS [108]. However, in this work we only compute exact inner products.

As the retriever \( f_{\text{ret}} \), we experiment with two choices: P-SP [289] and BM25 [220]. We implement BM25 using the retriv library from [14]. In order to normalize and calibrate BM25 scores, we compute the F1-score unigram token overlap [213] between the candidate \( y' \) and the best retrieval \( y^* \) to get a detection score in \([0, 1]\).

### 5.5.2 Controlled comparisons of retrieval with other AI-generated text detectors

First, we conduct a controlled comparison between the detection algorithms evaluated in Section 5.4.3 and our retrieval method on long-form question answering.\(^\text{12}\) We construct

---

\(^\text{12}\) Corresponding results in open-ended text generation on Wikipedia prompts are provided in Appendix D.4.
Table 5.3: A comparison of retrieval against other detectors on long-form QA (300 generated tokens). Our detector outperforms baselines (at 1% FPR) even with the most diverse paraphrases (+60L,O).

two retrieval corpora for this experiment: (a) a corpus of 3K sequences generated by a specific LM for one of the tasks; and (b) a corpus of 9K sequences formed by concatenating the generations from all three LMs considered in this chapter. We expect (b) to be a more difficult test for our method than (a), since the retriever needs to distinguish between multiple generations from different models given the same prompt. Next, we perform retrieval over this corpus using different types of queries: the original AI-generated text, its DIPPER paraphrase, and human-written text (each query with at least 50 tokens).

Table 5.3 shows that **across all LMs, retrieval is a much more effective detector than baseline detectors.** On unperturbed AI-generated text, retrieval has a 100% detection accuracy due to exact match with the retrieval corpus. On paraphrased text, retrieval with BM25 is quite effective, detecting 97.8% of the highest-diversity paraphrases (L60, O60) on GPT2-XL, 97.3% on OPT-13B and 96.2% on GPT-3.5 in long-form question answering. This is significantly better than the next best alternative with competing detectors (55.8%, 51.4%, 38.1%). Even on our harder augmented database of 9K generations, detection rates continue to be high (95.2%, 94.4%, 96.0%). Finally, we observe that BM25 is a more effective retriever than P-SP, scoring 95.2% vs 75.4% on the augmented setting in GPT2-XL.

These trends are consistent across different FPR thresholds, as shown in Figure 5.5.
Figure 5.4: Detection rate using retrieval at 1% FPR w.r.t. corpus size (left) and query length (right).

5.5.3 **Is retrieval an effective detector with a large retrieval corpus?**

In the previous section, we conducted experiments using the set of 9K sequences generated by all three models as the retrieval corpus. However, this is more of a toy experiment: in practice, a popular LLM API may serve millions of queries a day. As the corpus grows larger, the false positive rate (i.e., human-written text falsely detected as AI-generated) will grow. How well do retrieval-based detectors scale? To answer this question, we need access to a large corpus of AI-generated text. We utilize the training data used to train RankGen [129], which contains over 70M AI-generated sequences. We use the Project Gutenberg and Wikipedia splits of the training data, each of which contain 15M sequences generated by a T5-XXL model [211] fine-tuned on the different documents in the same domain. We discard generations which are shorter than 50 tokens, and paraphrase a subset of 2K generations to evaluate retrieval.

**Retrieval is effective even with a corpus size of 15M generations.** In Figure 5.4a, we plot the detection accuracy as a function of retrieval database size. Overall, we observe that
detection accuracy remains consistently high across different corpus sizes (varying from 1M generations to 15M generations). We observe slight drops in performance as the corpus size increases: just 1% (98.3 to 97.3) on Project Gutenberg (PG19) and 9.6% (90.0 to 80.4) on Wikipedia. Consistent with the results in Section 5.5.2, BM25 continues to outperform P-SP across different corpus sizes.

**Retrieval detection works best with 50 or more tokens of generated text.** Another important factor for our retrieval-based detector is the query length: shorter queries are likely to have more matches (many of them spurious) compared to longer ones. In Figure 5.4b, we plot the detection accuracy of paraphrased sequences at various query lengths by truncating each sequence to its first $X$ words before using it as a query for BM25. We use a retrieval corpus of 2M generations for this experiment. We observe that BM25 struggles to detect
paraphrased text with a query length of 20 (less than 25% accuracy), but the detection rate rapidly increases and begins to plateau at 50 tokens.

5.5.4 Ideas to make retrieval detection work well at an even larger scale

In Section 5.5.3, we observed that our proposed retrieval detector is effective even with a large corpus of 15M previously-generated sequences. While we do not have access to a larger corpus of generations (billion-scale), in this section we describe some ideas to improve retrieval detection at such a scale.

1. Timestamp filtering in retrieval corpus. To reduce the large search space, the detector interface could provide users with an option to restrict retrieval to only a fixed time period during which the text was likely to be generated. For instance, a common use-case of AI-generated text detection might be when teachers attempt to catch plagiarism in college essays. Teachers could restrict retrieval to only those generations created during the assignment window.

2. More sophisticated retrieval strategies. In our work, we only explore simple retrieval strategies like BM25. However, several more sophisticated retrieval strategies exist, which are known to boost performance [249] and could be useful here. These include methods like re-ranking of top-$k$ retrievals [120] or dense retrieval [114]. We do note that these more complex methods are also slower, and latency is likely to be a pressing concern for API providers.

3. Fine-tuning dense retrievers for the detection task. The retrievers in our work are not fine-tuned for the task of AI-generated text detection. However, we hypothesize that fine-tuning retrievers on this task can help retrievers adapt better to the retrieval corpus and detection task. Specifically, a contrastive learning approach could be adopted here: positive pairs are paraphrased or otherwise noised sequences paired with their generations, while negative pairs are human-written continuations paired with the machine-generated text.
5.5.5 Limitations of retrieval for detection

While retrieval over previously-generated sequences is an effective defense against paraphrase attacks, it also suffers from key limitations, some of which apply broadly to all existing detectors. We discuss these limitations below and discuss possible solutions:

1. **Detection is specific to an API.** Unlike other general-purpose AI detection algorithms e.g. OpenAI’s classifier [194], retrieval can only detect generations from the API over which the database is built. API #1 has no access to the database of generations from API #2, and thus will not be able to detect generations produced by API #2.

2. **The API provider needs to provide a retrieval infrastructure.** After the release of ChatGPT [225], AI chatbots are getting widespread adoption. At a conservative rate of 5M queries a day, the database will have almost two billion entries in a year. Complex retrieval infrastructure (like modern search engines) will be necessary to retrieve over these large databases with low latency.

3. **False positives due to training data memorization.** Language models have been shown to memorize sequences verbatim from their training data [25], such as the Gettysburg Address [208]. Despite being originally written by humans, these sequences will be classified as model-generated by our detector. To tackle this issue, we suggest API providers additionally perform retrieval over the training data used to train the model. If a sequence is found in the training set as well as the generation database, it is likely to be an instance of training set memorization.

4. **Privacy concerns.** Providing a retrieval detection service partially exposes the database of previously generated text by *all* users. This raises concerns of membership inference attacks [236] on private user data which may appear in the generated text. To mitigate this, we suggest: (1) users should be encouraged not to provide any sensitive private data
in their prompts to APIs, a practice already followed by ChatGPT\textsuperscript{13} and Bard\textsuperscript{14}; (2) API providers only provide a binary output from this detector (AI-generated or not), rather than actual search results; and (3) API providers rate-limit queries from IP addresses.

5. \textbf{Slight reduction in accuracy with large databases.} As we observed in Section 5.5.3, the accuracy of detecting paraphrased text slightly degrades as the database of retrievals gets larger. However, we found this decrease to be quite small (only 1% on PG19 scaling 1M generations to 15M), despite using fairly primitive retrievers like BM25. Moreover, unperturbed AI-generated text will always be detected with 100% accuracy using our method, irrespective of corpus size.

6. \textbf{Tasks with constrained output space or short outputs.} Similar to all other detection algorithms, it may be hard or even impossible to distinguish AI-generated outputs for tasks with a constrained output space (like sentence-level translation, classification) or very short outputs (as shown in Section 5.5.3). Thus, we believe the main utility of AI-generated text detection is for longer-form generated text, and hence we focus on tasks like long-form QA and open-ended text generation with relatively lengthy outputs. Note that to avoid detection, a sophisticated attacker may try to generate long-form text in smaller chunks using multiple API calls, where each newly-generated chunk is incrementally concatenated to the prompt. This is not a concern for our method if retrieval is done over the corpus of prompts concatenated with generations.

7. \textbf{Iterative attacks with access to detector.} A final concern is that attackers with access to detection algorithms will iteratively modify their perturbations until they avoid detection. While this is a valid concern for all detectors, we believe retrieval has an important advantage over the alternatives. Since the corpus of previously-generated text is proprietary, only the API provider can provide access to this detection service - it is

\textsuperscript{13}\url{https://chat.openai.com}
\textsuperscript{14}\url{https://bard.google.com}
impossible for attackers to locally reproduce this detector. This allows API providers to adopt several mitigation strategies such as (1) rate-limiting queries to avoid iterative attacks; (2) providing retrieval access only to verified users (e.g., teachers); and (3) detecting possible iterative attacks by analyzing previously queries to the retriever.

5.6 Experiments measuring intrinsic paraphrase generation quality

Our experiments in Section 5.4 and Section 5.5 focused on attacking AI-generated text detectors with paraphrases and defending against these paraphrase attacks. We used DIPPER as the underlying paraphrase generation model for all of these experiments. Are DIPPER’s paraphrases actually good enough to make the attack worthwhile, and can simpler paraphrasers be just as effective as DIPPER? In this section, we conduct careful ablation experiments (Section 5.6.1) and human evaluations (Section 5.6.2) to validate the effectiveness of DIPPER at preserving the semantics of the input generation. Our results show that DIPPER effectively leverages surrounding context to paraphrase multiple sentences while preserving input semantics.

5.6.1 Ablation studies on DIPPER

In this section, we perform automatic evaluations to confirm the efficacy of DIPPER as a paraphraser. From a survey of existing paraphrasers that we carry out in Appendix D.2.1, DIPPER possess two unique features that differentiate it from other paraphrasers: (1) its ability to leverage context from outside of the text to be paraphrased (such as the prompt); and (2) its ability to paraphrase multiple sentences at once. How useful are these features while paraphrasing long sequences of text?

To answer this question, we first train an ablated version of DIPPER by constructing a training dataset (Section 5.3) without any left or right context, and then fine-tuning T5-XXL using the same hyperparameters as in Section 5.3. We call this model DIPPER-no-ctx. We paraphrase 1K open-ended generations from GPT2-XL using both DIPPER and DIPPER-no-
Open-ended generation with GPT2-XL on Wikipedia prompts

<table>
<thead>
<tr>
<th>Control</th>
<th>Rewrite A</th>
<th>Rewrite B</th>
<th>GPT3.5 davinci-003 PPL</th>
<th>Rewrite A</th>
<th>Rewrite B</th>
<th>Unigram overlap with prompt</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RANKGEN-XL</td>
<td>GPT3.5 davinci-003 PPL</td>
<td>unigram overlap with prompt</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>rewrite A</td>
<td>rewrite B</td>
<td>rewrite A</td>
<td>rewrite B</td>
<td>rewrite A</td>
<td>rewrite B</td>
</tr>
<tr>
<td><strong>Experiment 1</strong>: Is context helpful for paraphrasing?</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rewrite A = DIPPER with context</td>
<td>rewrite B = DIPPER no context</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20L</td>
<td>65% 10.2</td>
<td>35% 9.2</td>
<td>71% 11.5</td>
<td>29% 12.6</td>
<td>55% 41.3</td>
<td>45% 40.7</td>
</tr>
<tr>
<td>40L</td>
<td>64% 9.8</td>
<td>36% 8.5</td>
<td>70% 11.9</td>
<td>30% 13.0</td>
<td>57% 40.7</td>
<td>43% 39.9</td>
</tr>
<tr>
<td>60L</td>
<td>67% 9.6</td>
<td>33% 7.6</td>
<td>68% 12.3</td>
<td>32% 13.6</td>
<td>56% 39.9</td>
<td>44% 39.2</td>
</tr>
<tr>
<td>60L,60O</td>
<td>65% 8.3</td>
<td>35% 6.4</td>
<td>75% 12.9</td>
<td>25% 15.0</td>
<td>58% 39.4</td>
<td>42% 38.2</td>
</tr>
</tbody>
</table>

| **Experiment 2**: Is it helpful to paraphrase multiple sentences at a time? |
| rewrite A = DIPPER 3 sentences at a time | rewrite B = DIPPER 1 sentence at a time |
| 20L      | 58% 9.2   | 42% 8.6   | 86% 12.6               | 14% 15.3  | 48% 40.7  | 52% 40.9 |
| 40L      | 56% 8.5   | 44% 8.1   | 83% 13.0               | 17% 15.8  | 45% 39.9  | 55% 40.4 |
| 60L      | 54% 7.6   | 46% 7.5   | 79% 13.6               | 21% 15.7  | 45% 39.2  | 55% 39.9 |
| 60L,60O  | 50% 6.4   | 50% 6.4   | 85% 15.0               | 15% 19.6  | 42% 38.2  | 58% 39.5 |

| **Experiment 3**: Does paraphrasing preserve the quality of the original text? |
| rewrite A = no paraphrasing | rewrite B = DIPPER |
| 20L      | 50% 10.4  | 50% 10.2  | 61% 11.1               | 39% 11.5  | 51% 41.6  | 49% 41.3 |
| 40L      | 57% 10.4  | 43% 9.8   | 67% 11.1               | 33% 11.9  | 55% 41.6  | 45% 40.7 |
| 60L      | 58% 10.4  | 42% 9.6   | 73% 11.1               | 27% 12.3  | 58% 41.6  | 42% 39.9 |
| 60L,60O  | 68% 10.4  | 32% 8.3   | 79% 11.1               | 21% 12.9  | 61% 41.6  | 39% 39.4 |

Table 5.4: Ablation experiments demonstrate the high quality of DIPPER’s paraphrases compared to alternatives. Displayed scores are the percentage of cases in which rewrite A is preferred over B by one of the three metrics, with subscripts showing absolute average scores on each metric across the dataset. Overall, DIPPER benefits from context outside the input (Experiment 1), multi-sentence paraphrasing (Experiment 2), and is not too far behind non-paraphrased text in terms of quality (Experiment 3).

cctx, using each of the four configurations of diversity control codes studied in this paper. We then evaluate the quality of the paraphrased text using three metrics: (1) GPT3.5-davinci-003 perplexity [23] of the prompt concatenated with the paraphrased continuation; (2) RANKGEN compatibility between the prompt and the paraphrased continuation [129]; and (3) unigram token overlap between the paraphrased continuation and the prompt.
Contextual paraphrasing leads to higher quality paraphrases. In Table 5.4 (Experiment 1), we observe that across all four control code configurations and all three metrics, paraphrases from DIPPER are preferred over paraphrases from DIPPER-no-ctx. Specifically, with the lexical and order control codes set to 60% (most diverse), DIPPER paraphrases are preferred by GPT3.5 perplexity 75% of the time compared to non-contextual paraphrases (average perplexity drop of 12.9 vs 15.0).

Paraphrasing multiple sentences at a time is better than paraphrasing individual sentences. Next, we use our DIPPER-no-ctx model to compare two settings: paraphrasing 3 sentences at a time vs paraphrasing 1 sentence at a time before concatenating. We hypothesize that the former will produce higher quality paraphrases since we expect it to better connect discourse elements across the text. Indeed, in Table 5.4 (Experiment 2) across all control codes, GPT3.5 and RANKGEN usually prefer multi-sentence paraphrases over the single-sentence baseline. This preference is 79% or higher for all control codes when evaluating with GPT-3.5 perplexity, reaching 85% for L60,O60.

DIPPER paraphrases are close to the unperturbed GPT-2 XL generations. Finally, we compare DIPPER with the original GPT2-XL generations (without paraphrasing) on the same three metrics. While we expect metrics to prefer non-paraphrased text, a strong paraphraser will produce text that is close to the original in terms of these metrics. Table 5.4 (Experiment 3) confirms our hypothesis: at L20, RANKGEN has a 50-50 preference between the two outputs, while GPT3.5 prefers the non-paraphrased generations just 61% of the time, with an average perplexity gain of just 0.4 (11.1 to 11.5). At more diverse control codes, preference for GPT2-XL generations does go up (58% RANKGEN, 73% GPT3.5 for L60), but absolute scores continue to be close (11.1 vs 12.3 GPT-3.5 perplexity). Note that while all of these ablations use just a single paraphrase sample, it is easy for an attacker to obtain multiple samples from DIPPER and choose the sample that maximizes these metrics (as discussed in Section 5.4.3).
Table 5.5: This table shows how often each point in the Likert scale was chosen by 3 annotators for the pairs of original and paraphrased texts. Twenty text pairs are randomly selected for each lexical code (L). 81.8% of the time, our model DIPPER provides a paraphrase which is nearly equivalent to the input in terms of semantic meaning.

5.6.2 Human evaluation of semantic preservation using DIPPER

The automatic semantic similarity scores in Table 5.1 and 5.2 indicate that DIPPER generates paraphrases that are faithful to the original input paragraphs. To confirm this result with human evaluation, we hire three native English teachers and/or editors on Upwork\(^{15}\) to evaluate the semantic fidelity of the paraphrases. As human evaluation is expensive, we fix the order diversity ($O$) to be 0 and focus on the impact of the lexical diversity. We evaluate paraphrases with the lexical codes $L_{20}$, $L_{40}$, and $L_{60}$, corresponding to moderate, medium, and high lexical diversity. Twenty paraphrases are sampled randomly for each lexical code, resulting in 60 original text and paraphrase pairs.

The evaluation is conducted on the platform Label Studio [256].\(^{16}\) As shown in the interface of our annotation platform Figure 5.6, the text to be paraphrased (highlighted in yellow) are preceded by its context. The annotators see the same amount of text as DIPPER. They need to first read the texts, select one point on the Likert scale, then provide free-form comments justifying their ratings. We estimated that the evaluation of each paraphrase takes 1.5 to 2 minutes. As such, we pay $15 as a base rate with a bonus for the reasonable extra time that the annotators spend on the tasks.

\(^{15}\)https://www.upwork.com

\(^{16}\)https://labelstud.io/
Among the 60 original text and paraphrase pairs, the three annotators agreed on their choice 28.3% of the time, and 60% of the time the point they chose on the scale differs by 1. Table 5.5 reports how often each point on the Likert scale is chosen. Over 80% of the time, our annotators rate DIPPER’s paraphrases as nearly equivalent (4 out of 5) or approximately equivalent (5 out of 5).

A qualitative analysis of the free-form annotator comments reveals systemic strengths and shortcomings of DIPPER. Table 5.6 provides two representative examples for each lexical code that is evaluated in our human study.

**Strengths** First, the third example in Table 5.6 exemplifies DIPPER’s ability to leverage information from context to increase diversity while maintaining coherence (i.e., from
line... reference the song’s title to reference to “I’m the Greatest”). The same is observed in row 2 where DIPPER uses the context to interchange he and Churchill. A paraphrase model without looking into context will have great difficulty in doing this and no prior paraphraser (see Table D.1 for a list) is capable of that. Second, the example in the fifth row highlights DIPPER’s ability to make significant changes to original texts with a high lexical diversity code (L60) (see the color coding) while preserving their semantic meaning as rated by the annotators.

**Qualitative shortcomings:** The first shortcoming is that, when the original text contains new created proper names (unlike common people and country names), such as the ones in row 6 (Homing Attack and Slide Attack), a high lexical code has a tendency to change such nouns, leading to the result that one of our annotators deems it to be only topically related to the original. However, this shortcoming can be overcome by decreasing the lexical code, which a user can choose from a continuous range (from 0 to 100). For instance, in row 1 with \texttt{lex}=20, the songs’ names \textit{M’s Confession} and \textit{Gone Fishing} are kept intact. Another shortcoming is that DIPPER occasionally omits content from an original text. While in some cases such removal is acceptable (see row 6), in other cases it causes significant change in the meaning of the text (see row 4). However, the former case can be overcome by paraphrasing a shorter paragraph at a time.

Overall, the human study shows that DIPPER performs well at preserving the semantic meaning of original texts while introducing both semantic and syntactic diversity. Because DIPPER provides user-friendly controllability of output diversity, a user can adjust the control code to find the most suitable paraphrase for their need.

### 5.7 Conclusion

We present DIPPER, a controllable paraphraser that can rewrite paragraphs in context. We use DIPPER to stress test current AI-generated text detectors, and we find that DIPPER paraphrases easily evade these detectors while preserving input semantics. As a defense, we
propose a simple retrieval-based detector which searches through a corpus of previously-generated sequences from an LLM API for semantically-similar generations to a given query. We show that this defense significantly outperforms baselines on paraphrated text, and scales effectively. In Appendix D.1 we discuss the ethical considerations of our work. We have open sourced our models, code and data to enable future research.\textsuperscript{17}

**Impact / Retrospective**

This study was originally released as a preprint on arXiv at the end of March 2023 [132]. In the last four months, DIPPER is receiving over a thousand downloads every month on the HuggingFace Hub\textsuperscript{18}, and has been used in over thirty follow-up studies. A number of these follow-up studies have used DIPPER to stress test their AI-generated text detectors [223, 169, 122], used our datasets [316, 259], and closely followed our evaluation protocol [235, 304]. The GPTZero team has also suggested that motivated by our work, they are building retrieval-based defenses in their product.\textsuperscript{19}
Table 5.6: Representative model outputs of each lexical code with Likert ratings and comments from the annotators. The texts in bold in the original texts are the context. Red words are the content being changed in the original text and green words are the changed content in the paraphrases.
So far this thesis described some critical issues in current LLMs, and my research aimed at building algorithms to make progress towards solving these issues. Looking ahead, my overall research goal is to build long-form text generation systems which are safe, interpretable, and capable of complex reasoning. To achieve this, I firmly believe that algorithmic development is needed in every part of the LLM development pipeline - data curation, architecture, pre-training, instruction tuning, and evaluation. This concluding chapter will describe some future directions that I am most excited about, which would be needed to make progress towards this vision.

6.1 Plan-based long-form generation

I am excited about training LLMs which can perform complex reasoning by generating plans of reasoning rather than the final responses directly. To tackle issues of hallucination in LLMs, recently several commercial tools like Perplexity AI have been released which combine search engines with LLMs, similar to our approach in Chapter 2. While these models have high accuracy on simple knowledge-intensive tasks, they struggle on complex queries such as “How well connected are airports near Amherst to the UK”. I hypothesize that these systems heavily rely on the direct presence of information on the internet, and are less capable of compositional multi-step reasoning [49]. As most of our models are trained
on US-centric data, without strong reasoning capabilities they will be unable to generalize well to tail-distribution entities (Amherst → Nagpur, UK → Middle East).

My vision is that we should strive to build LLMs which will generate a “plan of action” rather than answer questions directly. In the previous example, a plan-based system would create an interpretable script which will a) look for airports near the cities, filtering domestic airports; b) use mapping tools to understand distances to nearest airports; c) check latest flight schedules before presenting users with the answer. Unlike recent methods like chain-of-thought prompting [283], a plan-based LLM will be trained to discover reasoning chains zero-shot [124], rather than be shown close reasoning demonstrations during inference time. Moreover, I believe these reasoning steps should be tightly integrated with external APIs and multi-modal data.

I am excited about recent work taking the first steps in this direction [119, 224, 9]. One way to achieve this goal could be prompting large language models to automatically create this kind of training data using methods like Self-Instruct [280]. This data could then be filtered to keep only those examples with correct reasoning chains. As test-beds, geospatial problems (such as those tackled by Google Maps), mathematical reasoning, and reasoning tasks over scientific literature [43] are some interesting problems to look at, which often have deterministic answers.

6.2 Secure long-form generation

As discussed in Chapter 5, there is a growing risk of misuse and privacy issues with the widespread deployment of LLMs. Besides the topics discussed in Chapter 5 (AI-generated text detection), I believe there are several emerging problems in this space which need urgent research attention. For instance, there is a growing fear that LLMs can reproduce their training data verbatim, sparking privacy concerns [25]. While differential privacy is a possible solution, it often degrades accuracy [2, 11] and it is unclear if alternatives exist. Another concerning issue that has emerged after the release of ChatGPT, is that of
prompt injection. Certain adversarial inputs can encourage an LLM to ignore previous instructions [68], or write toxic / biased text [271]. Finally, another risk that has recently emerged due to the proprietary nature of LLMs is that of model theft (or model extraction). LLM training and development is a very expensive process, often costing many millions of dollars. These LLMs are often released behind pay-per-use black-box API services (like ChatGPT). This has exposed LLMs to an attack where an attackers attempt to reconstruct a local copy of a model with just black-box API access to it [103, 133]. Model extraction has recently emerged as a realistic issue, with there attempts [7] to reconstruct open-source versions of ChatGPT by training on its outputs.

I wish to build defenses against these attacks in my future research, and explore the usage of text anonymization and paraphrase generation systems for privacy in LLMs.

6.3 Understanding LLM training dynamics

In order to achieve my vision of plan-based and secure long-form generation, I firmly believe that more research in understanding LLM training is necessary. While clever prompt engineering techniques [283] are fascinating methods to push current LLMs towards this vision, I believe careful research in the entire LLM development lifecycle (data, pretraining, alignment, evaluation) is necessary to make substantial progress towards these goals. There are several interesting research questions in this space which I am excited about: 1) Can we intelligently choose our pretraining data based on some desired final requirements? 2) Can data selection and filtering be automated using reinforcement learning? 3) how well does instruction tuning / RLHF work in the pretraining phase? 4) what are the best pretraining objectives / transformer architectures to scale pretraining to long context windows of 100K tokens or beyond? 5) can retrieval / API usage be taught to LLMs during pretraining itself? 6) What is the relationship between optimization and the Chinchilla scaling laws [88]?
6.4 Evaluating recall in long-form text generation

Our study in Chapter 4 focused exclusively on the human evaluation of faithfulness or factuality in long-form summarization. However, faithfulness focuses exclusively on precision, and is just one of the aspects of long-form generated text which need to be evaluated. Another equally important aspect is recall or relevance. In other words, does the long-form generated output contain the important information needed to answer the input prompt?

Evaluating recall is a challenging problem, and requires knowledge of the pool of possible gold-standard facts which answers an input prompt. However, even within this pool of facts, a notion of relative importance of individual facts is needed to determine whether a model is generating the most critical information.

Assuming access to several gold-standard reference outputs, I hypothesize that automated versions of the Pyramid method [191] could be used to estimate the salience of different gold-standard facts. Modern large language models are successful in automatically extracting atomic facts from long-form text [182, 111, 295], and I believe can also be used to map atomic facts across references, to build the Pyramid.

Having access to several gold-standard reference outputs for input prompts in the wild is difficult in practice. However, in some domains like the biomedical NLP, there is a well-defined notion of salient information — PICO units are the most critical [43]. More broadly, I hypothesize that it may be possible to automatically extract guidelines for different prompt types, which specify the nature of the desired output. For example, prompts of the form “Tell me a biography of person X” could have a guideline like the following “include information about birth, nationality, profession and famous career achievements at the very minimum”. A large language model may be able to infer these guidelines using the right prompting and a number of human-written biographies, not necessarily of the same person as the test-time prompt. These guidelines could then be used by the large language model to judge the salience of generated outputs.
6.5 Using fine-grained human evaluation in RLHF

A final direction that I am excited about is utilizing fine-grained human feedback to improve large language models. In Chapter 4, we found that using fine-grained human feedback can lead to higher inter-annotator agreement while evaluating long-form text generation. Could this kind of feedback be helpful in supervising large language models as well, using reinforcement learning with human feedback? A recent work [296] has shown that fine-grained human feedback can lead to sample efficiency in RLHF, and I am excited to push this direction further. Future directions in this vein include: (1) using fine-grained fact verification models like FActScore [182] as reward models in RLHF; (2) using natural language sentences as feedback to improve large language models.
BIBLIOGRAPHY


APPENDIX A

APPENDIX FOR “LONG-FORM QUESTION ANSWERING: PROGRESS AND CHALLENGES”

A.1 Dataset Statistics
We downloaded the ELI5 dataset [57] from the KILT Github repository (https://github.com/facebookresearch/KILT). The dataset has 272,634 training examples, 1,507 validation examples and 600 test examples (answers hidden). The dataset uses English.

A.2 Training & Model Details
All our models are developed and trained using TensorFlow 1.15 [1] and Tensor2Tensor [261]. Our implementations are based on the open-source codebases of REALM ¹ and the Routing Transformer. ² Similar to the REALM implementation, we use separate processes to run the retriever and generate training data (using a MIPS search). Since our retriever is frozen, we do not use the document index refresher available in their codebase.

Retriever: Our retriever is trained on 64 Google Cloud TPUs for a total of 4k steps and a batch size of 12288. We do early stopping on the validation data (with a smaller batch size of 512 due to smaller P100 GPU memory). Our model converges quite fast, reaching its best performance in 1.5k steps (in 43 minutes) and needing 103 minutes for the full set of 4k steps.

¹https://github.com/google-research/language/tree/master/language/realm
²https://github.com/google-research/google-research/tree/master/routing_transformer
Generator: Our generator is trained on 64 Google Cloud TPUs, for a total of 100k steps on the ELI5 training set. We use the pg19_local_cluster8k configuration available in the Routing Transformer implementation. Besides the default hyperparameters, setting 15% input, attention and ReLU dropout was critical to prevent overfitting on the training set. We use a learning rate of 5e-5. Our retrievals, questions and answers are truncated / padded to 288 subword tokens (using the PG19 subword tokenizer). We use a minibatch size of 128 QA pairs, which corresponds to 332k tokens per mini-batch (of which, the loss is computed over the last 288 answer tokens, or 37k total tokens). We do not compute loss over padded tokens, and use special symbols to separate different parts of the input context. We reverse the retrieved paragraphs in context since the model uses local attention layers, and we wanted higher ranked retrievals to appear closer to the answer tokens. Our models take about 30 hours to finish 100k steps (0.92 steps / second).

Attention Maps: We show the 2D plots of our generator’s attention maps in Figure A.1.

Hyperparameter Choices: We experimented with several different pretraining strategies (using Wikipedia), smaller model variants and hyperparameter choices manually in preliminary experiments. All these experiments performed quite poorly on ELI5, producing very short and sometimes incoherent responses. Finally, switching to a Routing Transformer model which was pretrained on a longform language modeling dataset (PG-19) significantly improved generation quality. Hyperparameters for this pretrained model (like hidden size / number of layers) were manually chosen with model capacity in mind. For our final experiments with this pretrained model we did not perform any hyperparameter search during training, primarily due to the expensive setup required to train the system. During inference, we tuned the nucleus sampling value from 0.0 to 1.0 in increments of 0.1, choosing the value with the best validation set performance. Our hyperparameter choices for contrastive learning on the retriever have been discussed in an ablation study in Appendix A.3.

![Figure A.1: Figures (from [222]) showing 2-D attention schemes for the sparse attention mechanism used in Routing Transformer. Lower layers pool in local information via sliding window local attention (Sub-figure A.1a) while upper layers gather global information for every token via clustering (Sub-figure A.1b).](image)
A.3 Ablation Study of c-REALM

One of our contributions is scaling up a distantly supervised objective for training retrievers on ELI5, originally described in [106]. This method uses in-batch negative sampling, making minibatch size a critical hyperparameter for better contrastive learning. We perform controlled experiments initializing our retrievers with REALM-CCNews [80] and varying batch size and keeping all other hyperparameters consistent. In Table A.1, we notice a steady increase in performance as minibatch size is increased, with the largest gains coming by doubling the batch size in [106] from 512 to 1024. Finally, in preliminary experiments we saw no benefit of more intelligent negative sampling schemes.

<table>
<thead>
<tr>
<th>Batch size</th>
<th>R-Prec</th>
<th>Recall@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALM (pretrained)</td>
<td>6.6</td>
<td>14.9</td>
</tr>
<tr>
<td>256</td>
<td>6.2</td>
<td>11.0</td>
</tr>
<tr>
<td>512 [106]</td>
<td>6.8</td>
<td>12.6</td>
</tr>
<tr>
<td>1024</td>
<td>11.5</td>
<td>21.0</td>
</tr>
<tr>
<td>12288 (Ours)</td>
<td>13.3</td>
<td>21.2</td>
</tr>
</tbody>
</table>

Table A.1: The effect of minibatch size on the validation performance of c-REALM. As a baseline, we also add the retrieval performance of the REALM pretrained model which is used as an initialization.

Next, we investigate the effect of initialization on the training of c-REALM. Unlike [106] who initialize their model with BERT, before training we initialize our retriever with a pretrained self-supervised retriever. As a baseline, we initialize our model with ICT, a weaker self-supervised retriever introduced in [151]. Both models are trained with minibatch sizes of 12228. In Table A.2, we notice a large improvement in performance when using a better initialization, confirming our design decisions.

<table>
<thead>
<tr>
<th>Initialization</th>
<th>R-Prec.</th>
<th>R@5</th>
</tr>
</thead>
<tbody>
<tr>
<td>REALM (pretrained)</td>
<td>6.6</td>
<td>14.9</td>
</tr>
<tr>
<td>ICT [151]</td>
<td>9.3</td>
<td>16.5</td>
</tr>
<tr>
<td>REALM [80]</td>
<td>13.3</td>
<td>21.2</td>
</tr>
</tbody>
</table>

Table A.2: The effect of initialization on c-REALM. As a baseline, we also add the retrieval performance of the REALM-CCNews pretrained model without any finetuning on ELI5.
A.4 Number of trainable parameters

In Table A.3 we present the number of trainable parameters in our model compared to baselines on the leaderboard. Our generator is slightly larger than the models used in prior work, but we utilize a smaller retriever due to the shared query and candidate encoders in REALM. Overall, our system has a similar total number of parameters as baseline models like RAG and BART + DPR.

<table>
<thead>
<tr>
<th>Model</th>
<th>Generator</th>
<th>Retriever</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>T5-base</td>
<td>220M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>BART</td>
<td>406M</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>RAG</td>
<td>406M</td>
<td>220M</td>
<td>15B</td>
</tr>
<tr>
<td>BART + DPR</td>
<td>406M</td>
<td>220M</td>
<td>15B</td>
</tr>
<tr>
<td>RT + c-REALM</td>
<td>486M</td>
<td>110M</td>
<td>15B</td>
</tr>
</tbody>
</table>

Table A.3: The number of parameters used by our model and baselines. Our generator is slightly bigger than other submissions on the leaderboard, but we use a smaller retriever with a similar sized index.

A.5 Generations from our System

More generations have been provided (along with retrievals, highlighted to show n-gram overlap) in the supplementary material (data) as HTML files. We also present a few samples in Table A.10.

A.6 Performance on overlapping QA

In Table A.4 we compare ELI5 performance on the overlapping and non-overlapping subsets of the validation data. Since we only have human annotations for 300 questions (making the non-overlap subset have only 53 samples), we present this analysis using the QQP classifier’s outputs as well and refer to those numbers in the main text.
<table>
<thead>
<tr>
<th>Split</th>
<th>Retrieval</th>
<th>Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RPrec</td>
<td>R@5 F1</td>
</tr>
<tr>
<td>QQP classifier (1.5k examples)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>overlap (43.6%)</td>
<td>17.0</td>
<td>25.8</td>
</tr>
<tr>
<td>not overlap (56.4%)</td>
<td>10.4</td>
<td>17.7</td>
</tr>
<tr>
<td>AMT evaluation (300 examples)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>overlap (81%)</td>
<td>14.0</td>
<td>20.0</td>
</tr>
<tr>
<td>not overlap (19%)</td>
<td>5.3</td>
<td>17.9</td>
</tr>
</tbody>
</table>

Table A.4: ELI5 performance difference (for the $p = 0.6$ model) between subsets of validation QA having a question paraphrase (overlap) and not having a question paraphrase (not overlap) in the training set. We see the overlap subset has much better retrieval performance and slightly better generation performance.

### A.7 Human Evaluation Setup

We conducted several A/B tests between variants of our model using human annotators. We asked a total of 20 participants for help who voluntarily agreed to help with the annotation process. Most participants were English-speaking graduate students in computer science. In every test, participants were shown a question along with two answers (generated by different systems) presented in a random order. They were then asked to choose which generation (1) answered the question better / which answer was more relevant to the question; (2) was more coherent / had less repetition; (3) was more factually correct. Since some annotators had a limited time, we asked them to prioritize question (1) over (2) / (3). Annotators were allowed to select “Tie” if they could not choose between the systems. We also permitted them to use search engines, but suggested restricting search to Wikipedia. We present all our results in Table A.9. We also interviewed some participants after the annotation process and discuss our findings in Section 2.3.4. Note that while these A/B tests help us understand which system is relatively better, they do not provide an absolute measure of performance [26] — annotators reported that there were cases where both answers were very good and other cases where both were very poor. This is a limitation of A/B testing.

### A.8 Effect of length on ROUGE-L

In this section we measure the effect of outputs lengths on ROUGE-L scores. To conduct this experiment, we truncate generations by our system to a fixed fraction of tokens across all instances. As we see in Table A.5 in the *Truncate* column, shorter generations tend have lower ROUGE-L. To disentangle the effects of length and content, we also measure the generation quality by repeating the truncated generations several times until it matches the original generation length. In the *Repeat* $1/f$ *times* column, we notice a gap between our model’s original generation (24.4 ROUGE-L) and the equal-length truncated generations with repetition. These results indicate that while length helps improve ROUGE-L scores, simple repetition is insufficient.
Table A.5: Effect of truncating generations (Truncate) from the $p = 0.6$ model to keep the first $f$ fraction of tokens, and then repeating the truncated generations $1/f$ times to match the original length (Repeat ...). Notice a consistent increase in ROUGE-L with longer outputs, but a gap between the original generations (24.4) and equal-length generations formed by repeating truncations (Repeat $1/f$ times column).

A.9 More experiments on measuring retrieval grounding of generations

In this section we provide some more experiments testing the grounding of generations in retrieved documents. Overall, trends are consistent with our observations in Section 2.3.1.

Scatter plots between generation quality and unigram overlap with retrievals: We present this scatter plot in Figure A.2. There is virtually no correlation between the two quantities, with Spearman $\rho = 0.09$.

Instances with correct predicted retrieval: In Table A.6, we present results similar to Section 2.3.1 considering only those instances where at least one retrieved document matched the gold annotation (roughly 23% instances). We also present a scatter plot on the same set of instances in Figure A.3 and note a low correlation of $\rho = 0.13$.

Experiments with $p = 0.9$: We conduct additional experiments studying our model variant with higher nucleus sampling values. As we saw in Section 2.2.3, these generations tend to be more fluent and coherent, but less relevant to the question. In Table A.7 and Table A.8 we find consistent trends as Section 2.3.1, with very little difference between models conditioned on retrievals from c-REALM and random retrievals.
Figure A.2: Scatter plot for generations from the $p = 0.6$ model between generative quality (ROUGE-L vs reference on X-axis) and grounding with retrieval (unigram overlap with retrieved documents on Y-axis). The plot shows no correlation between the two quantities.

Table A.6: Comparison of generations conditioned on retrievals from c-REALM (Predicted) and randomly chosen retrievals (Random), for those cases where c-REALM predicted the correct retrieval. Notice very small differences in generation quality (R-L) as well as the fraction of $n$-grams ($n$-g) in the generation overlapping with retrievals predicted by c-REALM (vs predicted retr.). To control for overlap due to stopwords, we also add $n$-gram overlaps with the randomly sampled retrievals.
Figure A.3: Scatter plot for generations from the $p = 0.6$ model between generative quality (ROUGE-L vs reference on X-axis) and grounding with retrieval (unigram overlap with retrieved documents on Y-axis). Unlike Figure A.2, this plot only considers those cases where C-REALM predicted the correct retrieval. The plot shows very little correlation between the two quantities (Spearman $\rho = 0.13$).

<table>
<thead>
<tr>
<th></th>
<th>vs predicted retr.</th>
<th>vs random retr.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R-L</td>
<td>1-g</td>
</tr>
<tr>
<td>Predicted</td>
<td>22.62</td>
<td>53.9</td>
</tr>
<tr>
<td>Random</td>
<td>22.56</td>
<td>53.1</td>
</tr>
<tr>
<td>Gold Ans</td>
<td>-</td>
<td>54.1</td>
</tr>
</tbody>
</table>

Table A.7: Comparison of generations (with $p = 0.9$) conditioned on retrievals from C-REALM (Predicted) and randomly chosen retrievals (Random). Notice very small differences in: (1) ROUGE-L vs gold answers (R-L); (2) $n$-gram overlap ($n$-g) with retrievals predicted by C-REALM (vs predicted retr.). Gold answers also have a similar overlap with predicted retrievals. To control for overlap due to stopwords, we also add $n$-gram overlaps with the randomly sampled retrievals.
Table A.8: A fine-grained version of Table A.7 measuring the unigram overlap of nouns/numbers in the generations with the input question (vs qn.), retrievals predicted by C-REALM (vs predicted retr.) and randomly sampled retrievals (vs random retr.). Similar to Table A.7, notice very little difference with and without retrieval.

<table>
<thead>
<tr>
<th></th>
<th>vs qn.</th>
<th>vs predicted retr.</th>
<th>vs random retr.</th>
<th>but not in qn.</th>
<th>but not in qn.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>9.1%</td>
<td>32.4%</td>
<td>12.0%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Random</td>
<td>9.4%</td>
<td>30.2%</td>
<td>12.3%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gold Ans</td>
<td>8.3%</td>
<td>28.8%</td>
<td>15.1%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.9: Human evaluations experiments with exact number of ratings shown in (·). Differences greater than 10% with more than 50 total samples have been bold marked. The experiments marked with ** have less than 50 samples, so it is difficult to draw meaningful conclusions.
Q: How do shampoo and conditioner work?

A: The key ingredients in shampoo and conditioner are usually surfactants, they tend to attract oil from your skin. Shampoo will usually contain an anti-dandruff agent (dandruff is caused by fungi in your skin which cause dandruff). They also contain some moisturizing agents (moisturizers are things that stay in your skin longer and protect your scalp from cleaning), and a lot of anti-dandruff (dandruff is a side effect of high doses of both surfactants and moisturizers) Conditioner will contain moisturizing agents, like a skin moisturizer. They tend to cause oils to be separated from your hair (e.g. wash your hair with conditioner instead of shampoo). They both have a foaming agent that makes them foam up (usually Hydrogen Peroxide). They also have a foaming agent that stops dirt from attaching to your hair. Some conditioner will also suppress the action of the shampoo (to help prevent dirt from attaching to your hair). So what’s the difference between shampoo and conditioner? Well, if you combine both the shampoo and conditioner

Q: How do shampoo and conditioner work?

A: Fire is the chemical reaction of fuel (oxygen) with (what we call) impurities (ash, soot, oil etc). These impurities are created due to a number of things. If you were to study the chemical composition of a log (or a candle) and analyse it you would discover the chemical formula (A, C, and G) which is a simplified version of what actually happens. There is always fuel, and there is always the ash/soot. Wood is made up of mostly carbon and hydrogen and these are the fuels that create the fire. Hydrogen and oxygen are reactive, and they form water (H2O) and CO2 (CO2). If we split a log into smaller pieces, we start to see the process of combustion as it occurs. The carbon is burned, and with the explosion of CO, H2O and H2CO2, we get water and carbon dioxide (mostly). If we smash a log together, we start to see what happens when we also hit another log with another log...imagine smashing two sand castles together. We see how the sand

Q: What goes on in those tall tower buildings owned by major banks?

A: The actual buildings are not there for profit but for show. There are many real world historical buildings that have little to no effect other than being a symbol of an economic entity or symbol of an era or location, and nothing more. For example look at Sears, Sears didn’t care what went on inside, it was all about the appearance of its location, the prestige of the location, the facilities and so on. It didn’t care about how long it took to operate, it was about how much people would pay to go see it. Sears was a landmark as a cultural movement and other big companies followed suit, so if you want to see a building you’ve never seen before, you have to go see Sears, just like you have to see a Toyota Camry for Toyota Camry. They used to be all about building new factories, some of them if I recall, but now that they’re bigger, that means that more factory jobs are coming to them. You’ve probably seen them in stores as stores where people buy and sell stuff, so there aren’t that many places for them to come from. Instead, it’s just for show, a symbol of rich people.

Table A.10: Example generations from our LFQA system with $p = 0.9$. 

<table>
<thead>
<tr>
<th>_generation</th>
<th>Example generations from our LFQA system with $p = 0.9$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A: The same thing that causes the clouds you see at the start of a storm to form in the first place. The air pressure changes as the jet moves through the air, causing it to cool down and condense into water droplets. These droplets are in fact a lot bigger than the water droplets in the clouds, so when the plane passes through them, they can cause the air pressure to change, causing the water droplets to condense into clouds. This is what you see when you look at a cloud formation from the ground. In addition to this, when the plane passes through the clouds, it also moves through the air, causing it to change its temperature, causing the water droplets to condense and form clouds. These clouds are often quite large, and can block out the sun. This is what you see when you look at a cumulonimbus cloud. These clouds are even larger than the water droplets in the clouds, and they block out the sun.</td>
</tr>
</tbody>
</table>

151
APPENDIX B

APPENDIX FOR “RANKGEN: IMPROVING TEXT GENERATION WITH LARGE RANKING MODELS”

B.1 More RANKGEN details

B.1.1 RANKGEN training details

We fine-tune the encoder of the T5 v1.1 models from [211] using large minibatches (see Table B.1 for sizes) on a Cloud TPU v3 Pod slice with 128 chips. Our models are implemented in JAX [22] using the T5X library [218]. Each model was fine-tuned for 100k steps, using a constant learning rate of 0.002 using the Adafactor optimizer [233].

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch Size</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>RANKGEN-base</td>
<td>4096</td>
<td>110.2M</td>
</tr>
<tr>
<td>RANKGEN-large</td>
<td>4096</td>
<td>342.3M</td>
</tr>
<tr>
<td>RANKGEN-XL</td>
<td>1536</td>
<td>1.2B</td>
</tr>
</tbody>
</table>

Table B.1: Minibatch size and number of trainable parameters across different RANKGEN variants. See Appendix B.5 for ablation studies justifying scale.

B.1.2 Implementation and timing details

In Figure B.1 we provided a simplified Python implementation (without minibatching) of our RANKGEN beam search algorithm. We implement this algorithm in two libraries — the first uses PyTorch with the popular HuggingFace Transformers library [293], which we test on a RTX 3090 GPU with 25GB memory. The second uses JAX [22] with the T5X library [218], and is tested on a single Cloud TPU v3 board with 32GB memory.1 While measuring decoding time for various hyperparameters (Appendix B.1.3.2), we focus on throughput [37], measuring wall-clock time after minibatching to the extent the hardware permits. We ensure consistent experimental settings across hyperparameters, using the same machine and making sure no other computationally expensive process is running on it.

1https://cloud.google.com/tpu/docs/system-architecture-tpu-vm#single_tpu_board
B.1.3 **RANKGEN** hyperparameter grid search

Our hyperparameter grid search is conducted on Wikipedia data with the smallest model considered (GPT2-medium), using MAUVE as our hill-climbing criteria. Our RANKGEN algorithm has three main hyperparameters — rerank length $L$, beam size $B$ and number of samples per beam $N$. The rerank length denotes the number of new tokens which are generated before a re-ranking step takes place. Number of samples denotes the number of generated sequences for each beam. The number of samples retained across different re-ranking cycles is the beam size (see Figure B.1 for exact implementation). Our RANKGEN grid search is conducted over the following configurations —

**rerank length** $L$: 5, 10, 20, 50, max_length tokens

**number of samples** (beam size $B$ * number of samples in every beam $N$):
- 1 sample — (1 * 1);
- 5 samples — (1 * 5);
- 10 samples — (1 * 10); (2 * 5);
- 20 samples — (1 * 20); (2 * 10); (4 * 5);
- 40 samples — (1 * 40); (2 * 20);

Additionally, we measure the extent to which full-length reranking works ($L = \text{max length}$, $B = 1$) by simply increasing the number of samples $N$ over-generated and then for re-ranking.

### B.1.3.1 MAUVE score tradeoffs

In Figure B.2 we study the MAUVE performance tradeoffs for different hyperparameter configurations for the GPT2-medium model evaluated on Wikipedia data. Overall, we observe —

- Across all hyperparameter configurations, RANKGEN significantly improves MAUVE score over a no re-ranking baseline.
- MAUVE scores improve for shorter rerank lengths, justifying the benefit of beam search over re-ranking of complete generations.
- For cases of full re-ranking (re-rank length = max length), increasing number of samples improves the MAUVE score (since RANKGEN has more generations to choose from), but improvements saturates after 60 samples (for both model sizes), with the largest gain from 1 to 10 samples.
- We find that rerank length = 20 with 20 samples (beam size 2, samples per beam 10) performs best across all configurations.

### B.1.3.2 Speed tradeoffs

In Figure B.3 we study the average time taken (in seconds) for a single generation on Wikipedia. Overall, in both our implementations we observe that —
Decoding a single sample is an order of magnitude faster than decoding multiple samples (“over-generation”), which is needed before any re-ranking with RANKGEN is possible.

Reducing the rerank length increases decoding time, since more generate / re-rank cycles are needed. These cycles cannot be parallelized since the generate and re-rank steps are dependent on each other.

Overall, we see observe that decoding time is roughly $O(BN/L)$, where $B$ is beam size, $N$ is the number of samples per beam and $L$ is rerank length. This is especially true for the T5X implementation.

We dig a little deeper into these numbers: is the extra compute time due to over-generation (generation of 10 or 20 samples instead of one) or RANKGEN re-ranking? In Table B.2, we measure the time taken to generate and score an individual instance. We see that re-ranking with RANKGEN takes only a fraction of the time (1-10%) compared to generation, which means that over-generation is the bottleneck. Also see Section 3.4.2 in the main body of the chapter for a performance / time tradeoff scatter plot.

<table>
<thead>
<tr>
<th></th>
<th>HuggingFace (GPT2)</th>
<th>T5X / seqio (T5)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>medium</td>
<td>XL</td>
</tr>
<tr>
<td>secs / gen</td>
<td>7.7e-1</td>
<td>2.9e0</td>
</tr>
<tr>
<td><strong>RANKGEN calls in same time as one generation</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base</td>
<td>108.5</td>
<td>408.5</td>
</tr>
<tr>
<td>large</td>
<td>42.8</td>
<td>161.1</td>
</tr>
<tr>
<td>XL</td>
<td>16.4</td>
<td>61.7</td>
</tr>
</tbody>
</table>

Table B.2: Number of RANKGEN calls in the same time as one LM generation. Across libraries and LM sizes, RANKGEN needs only a fraction of time vs generation.

### B.2 Human Evaluation Details

We hired freelancers from Upwork as well as two volunteers to perform our human evaluation. In total, our human evaluation had eight annotators. Following recent recommendations from [112], we ensured that each annotator (except one) was either an English teacher or an English writer. To avoid bias, we ensured that none of the annotators were computer science researchers, making them unaware of text generation research / RANKGEN.

---

2https://www.upwork.com
Setup: Annotators were shown a 200-250 word prefix, and were asked to choose one of two 80-100 word continuations. Annotators were not told which model generated each continuation, and we shuffled the continuations in a random order to avoid position biases (“blind A/B testing”). The job posting and instructions shown to the annotators are provided in Table B.14. We used Amazon Mechanical Turk Sanbox to collect our annotations, using the interface shown in Figure B.6. Note that we used the MTurk Sandbox interface only — no MTurk workers are recruited in our human study due to poor annotation quality for open-ended text generation [112, 32].

Screening: To ensure high annotation quality, we first asked annotators to complete a small screening test of 20 pairs with INBOOK distractors, keeping 80% accuracy as our passing criteria (estimated human performance on this set is 90-95%). We paid annotators 10$ for the screening test. Around half the interviewed Upworkers passed the test.

Main Task (comparing generations): In our main task comparing generations from RANKGEN with nucleus sampling, we asked annotators to choose the better continuation as well as provide a 1-3 sentence free-form explanation for their choice. We paid annotators 1$ for each pair, and provided a 10$ bonus at the end of a 100 pairs. Each annotator was provided with 100 instances (50 each from Wikipedia and PG19) either generated by the T5-XXL-C4 model [152] or GPT2-medium [208], with beam search outputs from RANKGEN-XL-all. Three annotators rate each model, giving us a total of 600 human annotations with explanations.

Main Task (INBOOK human estimate): Our second main task involved choosing the gold human-written continuation vs random INBOOK negatives. We paid annotators 0.5$ for this task, and did not ask them to explain their choices. This main task was similar in nature to our screening task.

---

3https://requestersandbox.mturk.com/
B.3 Suffix Identification

B.3.1 Gold vs INBOOK - Hard examples

In Section 3.2.1 and Appendix B.3.2 we make use of “hard negatives”. To select these harder negative from the document, we use a trained RANKGEN model (XL sized, trained on all four domains). Specifically, we use RANKGEN to score the compatibility of every 128-word token sequence in the document to the prefix, and take the highest scoring 10 sequences that are not the gold continuation (“Hard” negative). All negatives sequences start and end at sentence boundaries so that LMs cannot rely on local syntactic patterns. For our two-way classification experiments in Section 3.2.1, we consider a random sequence among these 10 hard negatives. Since RANKGEN-all-XL-both was used to find these hard negatives, results on this RANKGEN variant are not very meaningful (since they are adversarial to this variant by construction).
B.3.2 Gold vs INBOOK - more negatives

In Section 3.2.1, we used a single INBOOK to test models. How do models fare when they need to choose the gold continuation over multiple INBOOK negatives? In Table B.3 we perform experiments on a 11-way classification task (10 INBOOK negatives). Overall, we find that most LMs do barely above chance, whereas RANKGEN significantly outperforms large LMs (even GPT3).

<table>
<thead>
<tr>
<th>INBOOK neg type → Random</th>
<th>Hard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PG Wiki</td>
</tr>
<tr>
<td>Random</td>
<td>9.1  9.1</td>
</tr>
<tr>
<td>Unigram Overlap</td>
<td>42.3 18.5</td>
</tr>
<tr>
<td>GPT2-medium [208]</td>
<td>25.2 11.8</td>
</tr>
<tr>
<td>GPT2-XL [208]</td>
<td>28.2 12.4</td>
</tr>
<tr>
<td>T5-base (f.t. PG19)</td>
<td>28.8 14.3</td>
</tr>
<tr>
<td>T5-XXL (f.t. PG19)</td>
<td>38.8 17.5</td>
</tr>
<tr>
<td>T5-XXL-C4 [152]</td>
<td>34.3 14.6</td>
</tr>
<tr>
<td>GPT3 170B* [23]</td>
<td>32.0 14.0</td>
</tr>
<tr>
<td>RANKGEN (ours)</td>
<td>94.4 69.8</td>
</tr>
<tr>
<td>PG19-XL-INBOOK</td>
<td>45.0 28.5</td>
</tr>
<tr>
<td>PG19-XL-GENERATE</td>
<td>94.4 69.0</td>
</tr>
<tr>
<td>PG19-XL-both</td>
<td>92.6 84.6</td>
</tr>
<tr>
<td>all-XL-both</td>
<td></td>
</tr>
</tbody>
</table>

Table B.3: A version of Table 3.1 with 10 distractors (11-way classification). Like Table 3.1, large LMs perform poorly and close to chance on hard sets. *GPT3 scores computed using 100 datapoints. †The hard sets were adversarially constructed using this RANKGEN variant.

Gold vs all INBOOK negatives (“retrieval”): What if instead of 10 negatives, we used all possible INBOOK negatives in the book? This task could be framed as a retrieval problem akin to RELiC (Section 3.4.4): given a prefix, find the correct continuation from all possible continuations in the same book. Since PG19 books can be quite long, retrievers needs to search among 2538 candidates on average in the PG19 validation set. We present results on this retrieval task in Table B.4. Overall, we find that RANKGEN is quite successful at this task, getting a recall@1 of 48.2% with a model trained on just PG19 data and INBOOK negatives. Training on just PG19, increase model size, increasing minibatch size and using just INBOOK negatives helps improve retrieval performance. In initial experiments, we extensively used performance on this task to hill-climb and justify our design choices. Note that we do not test LMs on this retrieval task, since it is computationally expensive to do a forward pass for each of the 2538 candidates for each of the 100K datapoints.

B.3.3 Gold vs GENERATIVE - breakdown by generative model

See Table B.5 for a breakdown by the model used to create the GENERATIVE negatives.
<table>
<thead>
<tr>
<th>Model Size</th>
<th>Batch Size</th>
<th>R@1</th>
<th>R@3</th>
<th>R@5</th>
<th>R@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>4096</td>
<td>34.9</td>
<td>52.6</td>
<td>60.6</td>
<td>70.5</td>
</tr>
<tr>
<td>large</td>
<td>4096</td>
<td>45.2</td>
<td>62.8</td>
<td>69.9</td>
<td>78.1</td>
</tr>
<tr>
<td>XL</td>
<td>1536</td>
<td>48.1</td>
<td>65.4</td>
<td>72.1</td>
<td>79.7</td>
</tr>
<tr>
<td>XL-inbook</td>
<td>1536</td>
<td>48.2</td>
<td>65.5</td>
<td>72.1</td>
<td>79.7</td>
</tr>
<tr>
<td>XL-gen</td>
<td>1536</td>
<td>4.4</td>
<td>10.4</td>
<td>14.4</td>
<td>20.5</td>
</tr>
</tbody>
</table>

Table B.4: RANKGEN retrieval performance on PG19 validation books. On average, retrieval takes place over 2538 candidates. RANKGEN gets high performance on this task, and scaling model size, scaling minibatch size, training on just PG19 and using just INBOOK negatives improves recall@1 (R@1).

B.3.4 Details of Suffix Identification Datasets

ChapterBreak [244] is a 6-way classification task in which models are provided as input a long segment from a narrative that ends in a chapter boundary. Models must then identify the correct ground-truth chapter beginning from a set of negatives sampled from the same narrative — a task requiring global narrative understanding. ChapterBreak has two settings — (1) PG19 — the validation set of the Project Gutenberg language modeling benchmark [209]; (2) AO3 — a ChapterBreak split adapted from fan-fiction posted to Archive of Our Own (AO3). Although [244] provide prefixes up to 8192 tokens, we study ChapterBreak in the setting using just 256 tokens of prefix to ensure compatibility with the input lengths of RANKGEN. The ChapterBreak dataset is not divided into validation / test splits, so we simply use the single available split.

https://archive.org/download/AO3_story_dump_continuing
<table>
<thead>
<tr>
<th>Discriminator</th>
<th>GPT2-md PG19 wiki</th>
<th>GPT2-XL PG19 wiki</th>
<th>T5-XXL-PG19 PG19 wiki</th>
<th>T5-XXL-C4 PG19 wiki</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
<td>50.0</td>
</tr>
<tr>
<td>Unigram Overlap</td>
<td>38.4</td>
<td>43.6</td>
<td>36.7</td>
<td>39.8</td>
<td>42.3</td>
</tr>
<tr>
<td>GPT2-medium [208]</td>
<td>2.1</td>
<td>4.9</td>
<td>3.0</td>
<td>6.6</td>
<td>3.9</td>
</tr>
<tr>
<td>GPT2-XL [208]</td>
<td>12.7</td>
<td>23.3</td>
<td>1.7</td>
<td>4.6</td>
<td>9.9</td>
</tr>
<tr>
<td>T5-XXL (f.t. PG19)</td>
<td>46.2</td>
<td>54.6</td>
<td>23.5</td>
<td>29.7</td>
<td>36.6</td>
</tr>
<tr>
<td>T5-XXL-C4 [152]</td>
<td>24.7</td>
<td>52.2</td>
<td>10.9</td>
<td>26.1</td>
<td>29.1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RANKGen (ours)</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PG-XL-GENERATIVE</td>
<td><strong>96.9</strong></td>
<td><strong>91.4</strong></td>
<td><strong>95.7</strong></td>
<td><strong>88.8</strong></td>
<td><strong>91.9</strong></td>
</tr>
<tr>
<td>PG-XL-INBOOK</td>
<td>78.4</td>
<td>66.3</td>
<td>69.7</td>
<td>60.3</td>
<td>65.2</td>
</tr>
<tr>
<td>PG-XL-both</td>
<td>97.4</td>
<td>81.3</td>
<td>93.7</td>
<td>74.0</td>
<td>89.7</td>
</tr>
<tr>
<td>all-XL-both</td>
<td>94.3</td>
<td>84.5</td>
<td>88.8</td>
<td>78.0</td>
<td>83.5</td>
</tr>
</tbody>
</table>

Table B.5: A version of Table 3.2 breaking down performance by domain (Project Gutenberg PG19, Wikipedia) and model used to generate GENERATIVE negatives using nucleus sampling [89] with $p = 0.9$. Language model perplexity prefers GENERATIVE sequences over human text (as previously noted by [65]), especially when the GENERATIVE negative is generated by the same language model.

**HellaSwag** [307] is a 4-way classification task focusing on commonsense natural language inference. For each question, a prefix from a video caption is provided as input and a model must choose the correct continuation for this prefix. Only one out of the four choices is correct – the actual next caption of the video. HellaSwag is scraped from the video captions in ActivityNet [135] and how-to paragraph instructions on WikiHow. We study the setting where each of the 4 endings are complete sentences, which is constructed by prepending ctx_b to the given endings). We use the validation set of the HellaSwag corpus since the test set answers are hidden.

**StoryCloze** [185, 232] is a 2-way classification task designed to test commonsense reasoning. Systems are provided with the first four sentences of a five-sentence commonsense story, and must choose the correct ending to the story. We used the test set for the Spring 2016 split and the validation set for the Winter 2018 split (due to the hidden test set).
Table B.6: Zero-shot suffix identification results on existing datasets. RANKGEN significantly outperforms all LMs on ChapterBreak which has long prefix/suffix lengths. RANKGEN performs similar to similar-sized GPT2-XL on StoryCloze and HellaSwag, with shorter inputs and more local dependencies.

**B.3.5 RANKGEN for suffix identification**

RANKGEN is trained on a suffix identification objective: given a prefix, choose the gold continuation over INBOOK and GENERATIVE negatives. How well does RANKGEN learn this task? How does RANKGEN fare on existing suffix identification benchmarks?

**Performance on INBOOK / GENERATIVE:** In Section 3.2.1 we motivated the RANKGEN design by showing the inability of LM perplexity to prefer the gold continuations over negatives. How does RANKGEN fare on these negatives? In Table 3.1 and Table 3.2 we evaluate the performance at distinguishing gold continuations from negatives, and compare RANKGEN to large LMs. Since RANKGEN is directly optimized on this objective, it significantly outperforms large LMs (99.1% vs 78.2% with GPT-3 for INBOOK). RANKGEN variants trained on just INBOOK or just GENERATIVE perform best at their respective tasks, but we observe some generalization (INBOOK model gets 69.8% on GENERATIVE PG19 negatives, GENERATIVE model gets 80.2% on INBOOK negatives, both higher than all LMs). Strong performance on GENERATIVE could have several applications like fake news detection [308, 65], and is an interesting future work direction.
Performance on existing suffix identification benchmarks: We test RANKGEN on three existing suffix identification datasets — ChapterBreak [244], ROCStories cloze test [185] and HellaSwag [307]; dataset details are provided in Appendix B.3.4. To measure their intrinsic capability, models are evaluated zero-shot, without finetuning on training sets.5

In Table B.6 we find that RANKGEN significantly outperforms all LMs on ChapterBreak (64.3 vs 28.6). RANKGEN performs comparably to similar-sized GPT2-XL (1.5B parameters) on other tasks, beating it on StoryCloze (75.8 vs 72.2), but slightly worse on HellaSwag (46.3 vs 47.9). Much larger LMs like GPT3 170B [23] and PaLM 540B [31] perform best on StoryCloze and HellaSwag. Scaling also benefits RANKGEN (30.4 vs 40.7 on HellaSwag for base vs XL), and we believe further scaling RANKGEN is a promising direction for future work. We also find INBOOK negatives are more beneficial than GENERATIVE negatives (64.3 vs 33.6 on ChapterBreak PG19). We hypothesize that the different trends on different datasets can be attributed to input length. As seen in Table B.6, ChapterBreak has much longer inputs (240 prefix, 153 suffix tokens) than other datasets (35 prefix, 7 suffix tokens for ROCStories). The focus on local context in LMs [117, 231, 242] helps with short-range tasks but also likely contributes to their underperformance on complex long-range tasks like ChapterBreak.

B.4 More Evaluation Details & Results

B.4.1 MAUVE setup

We extensively use the MAUVE metric from [203] for automatic evaluation of our model. MAUVE is shown to have high correlation with human judgements of the quality of generated text. We closely follow the best practices listed in the official MAUVE repository,6 which we found critical in preliminary experiments. Specifically,

1. We ensure that each run has the exact same hyperparameters — using the default hyperparameters in the official MAUVE library.

2. We use 7713 generations per run, which is the size of our Wikipedia validation set. This follows the suggestion in the official codebase README of having at least 5000 generations for comparing models. While our PG19 validation set is much bigger, we truncate it to 7713 generations since MAUVE scores tend to reduce with more generations.

5[307] also describe zero-shot HellaSwag experiments, testing models on unseen WikiHow / ActivityNet categories; however they still finetune models on HellaSwag data for seen categories, while we do no such finetuning.

6https://github.com/krishnap25/mauve#best-practices-for-mauve
3. Since MAUVE scores are higher for shorter generations, we ensure that all tested methods have roughly equal generation lengths, between 70-80 words / 120-130 tokens. We also truncate human text / generations to ensure that each instance ends at a sentence boundary. In initial experiments we observed that truncating consistently for human text and machine text leads to lower MAUVE variation.

4. Due to variation in MAUVE score from run to run, we average the MAUVE score for nucleus / top-k / typical sampling over five runs. For the T5-XXL-C4 model on Wikipedia with nucleus sampling, the MAUVE scores were [0.803, 0.778, 0.759, 0.785, 0.768], giving a standard deviation of 0.015.

**B.4.2 MAUVE Divergence Curves**

The MAUVE metric is the area under a divergence curve, a curve which attempts to analyze the type of errors the model is making. Given \( P \) is the distribution of human text and \( Q \) is the distribution of machine-generated text, [203] describe two types of errors made by models —

**Type I**: \( \text{KL}(Q|P) \) — False positives, or cases where models generate text which is unlikely to be written by humans, like semantic repetitions common in neural text generators [89, 309].

**Type II**: \( \text{KL}(P|Q) \) — False negatives, or cases where models cannot generate text which is likely to be written by humans, sometimes seen with truncation strategies [226].

In Figure B.4 and Figure B.5 we plot the divergence curves comparing greedy decoding, nucleus sampling, and full sample re-ranking with perplexity and \text{RANKGEN}. We observe that re-ranking with \text{RANKGEN} increases the area under the curve, whereas re-ranking with model perplexity reduces the area. Re-ranking with \text{RANKGEN} reduces both Type I (bigger intercept on \( y = 1 \)) and Type II errors (bigger intercept on \( x = 1 \)). Re-ranking with perplexity leads to higher Type I errors, or more repetition (as also observed in Appendix B.4.3).

**B.4.3 Token Overlap metrics**

In addition to the MAUVE scores calculated in Section 3.3, we measure token overlap statistics comparing different decoding methods. First, we measure the \text{rep} metric from [286], which is an approximate measurement of the amount of repetition in generated text. We measure the percentage of generated tokens which are exactly copied from the immediate local prefix of 20 tokens. In Table B.7 we find that re-ranking with \text{RANKGEN} slightly reduces \text{rep} compared to nucleus sampling (18.9 vs 19.5). We get even lower repetition on the \text{RANKGEN} trained on just generative negatives (17.8), while \text{RANKGEN} trained on just inbook negatives gets 20.0 — thus generative negatives are better at reducing repetition. Re-ranking with perplexity increases \text{rep} to 23.9, whereas greedy decoding has the highest repetition of 59.5. This is consistent with recent findings of repetition in greedy decoded outputs [89, 309]. Human text is the least repetitive, with a \text{rep} score of 15.4.
Next, we measure the fraction of unigrams in the generation which are also present in the prefix. Higher scores could either imply more faithfulness to the prefix (less hallucination), or lower amounts of abstraction. We present two versions of this metric — (1) considering all tokens (Table B.8); (2) considering only only lemmatized nouns and numbers (Table B.9). Overall, we find that re-ranking samples with RANKGEN slightly increases this overlap score (19.5 vs 21.7), but re-ranking by token overlap (38.4) or perplexity (25.0) leads to a much higher score. Given the lower MAUVE scores for these two approaches (Table 3.3), we suspect that token overlap / perplexity re-ranking leads to lower amounts of abstraction / repetitiveness. Human written text has the lowest overlap, perhaps indicating more abstractive text.

### B.5 Ablation Studies

We conduct several ablation studies studying the importance of three aspects — (1) model size; (2) minibatch size, or number of negative samples during contrastive learning; (3) the type of negative samples (inbook, generative or both). Overall, we see clear benefits of increasing model size and increasing minibatch size for suffix identification (Table B.10, Table B.11) and human-text identification (Table B.13). We see a similar, but less prominent trend on MAUVE scores after re-ranking generations (Table B.12). For some settings we find that the RANKGEN-large variant produces slightly better generations than RANKGEN-XL. We hypothesize this is due to the much larger minibatch used to train RANKGEN-large models (4096) compared to RANKGEN-XL (1536) due to memory constraints.

### B.6 More Model Generations

More model generations with human explanations are provided in Table B.15 to Table B.20. See our attached data submission for all 600 annotations for the 200 generation pairs.
def rankgen_search(prefix, scorer, generator, rerank_length, beam_size, samples_per_beam):
    all_beams = ['']
    for _ in range(0, MAX_LENGTH, rerank_length):
        # concatenate input prefix with current beams
        all_inputs = [prefix + ' ' + beam for beam in all_beams]
        # for each beam, generate next rerank_length tokens.
        # samples_per_beam hypotheses are generated per beam,
        # making a total of (num_beams * samples_per_beam) hypotheses
        hypotheses = generator(all_inputs, num_new_tokens=rerank_length, num_samples=samples_per_beam)
        # measure RankGen score between prefix and each hypothesis
        scores = scorer(prefix, hypotheses)
        # take top-K scores where K=beam size
        top_indices = np.argsort(-1 * scores)[:beam_size]
        all_beams = [outputs[x] for x in top_indices]
    return all_beams

Figure B.1: A simplified Python implementation showing our RANKGEN beam search algorithm (without minibatching). For every rerank_length tokens, a generator suggests hypotheses and the RANKGEN scorer ranks them. The top beam_size hypotheses are retained for the next stage of generation and re-ranking.
Figure B.2: Variation in MAUVE score across different RANKGEN hyperparameters on Wikipedia data (Appendix B.1.3.1). **Left**: Experiments on GPT2-medium show that RANKGEN improvements are robust to hyperparameter choice, re-ranking shorter hypotheses improves performances over full re-ranking, re-ranking more samples improves performance. **Right**: Full re-ranking performance generally improves with more samples, but this improvement saturates after a point, especially for larger models (T5-XXL).

Figure B.3: Time taken (in seconds) for a single generation across different hyperparameter settings in both our implementations (HuggingFace / T5X). We see roughly linear increase in decoding time with number of samples, and linear increase with number of re-ranking steps (1 / rerank_length).
Figure B.4: Divergence curves [203] after full sample re-ranking on Wikipedia inputs using RANKGEN-XL trained on all four domains. The area under this curve is the MAUVE score. Overall, we see that RANKGEN makes fewer Type I (bigger intercept with $y = 1$ line) and Type II style errors (bigger intercept with $x = 1$ line). PPL re-ranking increases the amount of repetition in generated text (Table B.7), leading to more Type I errors (smaller intercept with $y = 1$ line).
Figure B.5: Divergence curves [203] after full sample re-ranking on PG19 inputs using RANKGEN-XL trained on PG19. The area under this curve is the MAUVE score. Overall, we see that RANKGEN makes fewer Type I (bigger intercept with $y = 1$ line) and Type II style errors (bigger intercept with $x = 1$). PPL re-ranking increases the amount of repetition in generated text (Table B.7), leading to more Type I errors (smaller intercept with $y = 1$).
<table>
<thead>
<tr>
<th>Decoding method</th>
<th>GPT2-md PG19 wiki</th>
<th>GPT2-XL PG19 wiki</th>
<th>T5-XXL-PG19 PG19 wiki</th>
<th>T5-XXL-C4 PG19 wiki</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Text</td>
<td>15.8</td>
<td>15.0</td>
<td>15.8</td>
<td>15.0</td>
<td>15.4</td>
</tr>
<tr>
<td>Greedy decoding</td>
<td>71.4</td>
<td>56.6</td>
<td>66.8</td>
<td>51.6</td>
<td></td>
</tr>
<tr>
<td>Nucleus, $p = 0.9$ [89]</td>
<td>21.8</td>
<td>18.8</td>
<td>22.4</td>
<td>19.5</td>
<td></td>
</tr>
<tr>
<td>Top-k, $k = 40$ [58]</td>
<td>19.4</td>
<td>17.0</td>
<td>19.9</td>
<td>19.7</td>
<td></td>
</tr>
<tr>
<td>Typical, $p = 0.9$ [176]</td>
<td>21.6</td>
<td>18.6</td>
<td>22.2</td>
<td>19.5</td>
<td></td>
</tr>
</tbody>
</table>

### Re-ranking 20 nucleus samples

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Unigram overlap</td>
<td>22.2</td>
<td>19.9</td>
<td>22.9</td>
<td>20.6</td>
<td>21.5</td>
</tr>
<tr>
<td>LM perplexity</td>
<td>26.9</td>
<td>23.2</td>
<td>27.9</td>
<td>24.3</td>
<td>24.6</td>
</tr>
<tr>
<td>RANKGEN PG-XL-gen</td>
<td>20.0</td>
<td>17.2</td>
<td>20.5</td>
<td>17.9</td>
<td>18.3</td>
</tr>
<tr>
<td>RANKGEN PG-XL-inbook</td>
<td>22.1</td>
<td>19.5</td>
<td>22.7</td>
<td>20.0</td>
<td>18.2</td>
</tr>
<tr>
<td>RANKGEN PG-XL-both</td>
<td>20.9</td>
<td>18.4</td>
<td>21.6</td>
<td>19.2</td>
<td>17.4</td>
</tr>
<tr>
<td>RANKGEN all-XL-both</td>
<td>20.5</td>
<td>18.6</td>
<td>21.1</td>
<td>19.4</td>
<td>17.3</td>
</tr>
</tbody>
</table>

Table B.7: Fraction of generated tokens which are copied from the previous 20 tokens, roughly measuring the amount of repetition in text (the rep metric from [286]). Overall we find that ranking samples with RANKGEN reduces repetition, whereas ranking with perplexity increases repetition. Greedy decoded outputs are the most repetitive, whereas human-written text is the least repetitive.

### Unigram overlap

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>33.6</td>
<td>43.5</td>
<td>34.4</td>
<td>45.7</td>
<td>39.9</td>
</tr>
<tr>
<td>LM perplexity</td>
<td>19.9</td>
<td>29.4</td>
<td>20.2</td>
<td>30.2</td>
<td>16.9</td>
</tr>
<tr>
<td>RANKGEN PG-XL-gen</td>
<td>18.8</td>
<td>25.5</td>
<td>19.3</td>
<td>26.5</td>
<td>14.6</td>
</tr>
<tr>
<td>RANKGEN PG-XL-inbook</td>
<td>18.8</td>
<td>25.1</td>
<td>19.4</td>
<td>26.4</td>
<td>15.9</td>
</tr>
<tr>
<td>RANKGEN PG-XL-both</td>
<td>19.4</td>
<td>25.2</td>
<td>19.7</td>
<td>26.5</td>
<td>15.7</td>
</tr>
<tr>
<td>RANKGEN all-XL-both</td>
<td>19.1</td>
<td>24.8</td>
<td>19.5</td>
<td>26.1</td>
<td>15.7</td>
</tr>
</tbody>
</table>

Table B.8: Percentage of unigrams in generation also present in the prefix. Overall, we see that re-ranking nucleus samples with RANKGEN increases this overlap, but not as much as re-ranking with LM perplexity. Human text has the lowest overlap, which we hypothesize is due to higher amounts of abstraction.
<table>
<thead>
<tr>
<th>Generator Language Model</th>
<th>GPT2-md PG19 wiki</th>
<th>GPT2-XL PG19 wiki</th>
<th>T5-XXL-PG19 PG19 wiki</th>
<th>T5-XXL-C4 PG19 wiki</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Text</td>
<td>19.6 27.3</td>
<td>19.6 27.3</td>
<td>19.6 27.3</td>
<td>19.6 27.3</td>
<td>23.4</td>
</tr>
<tr>
<td>Greedy decoding</td>
<td>23.8 31.1</td>
<td>23.0 30.5</td>
<td>21.8 26.2</td>
<td>26.5 33.2</td>
<td>27.0</td>
</tr>
<tr>
<td>Nucleus, $p = 0.9$ [89]</td>
<td>23.8 29.7</td>
<td>24.2 30.3</td>
<td>19.3 24.4</td>
<td>24.6 31.6</td>
<td>26.0</td>
</tr>
<tr>
<td>Top-k, $k = 40$ [58]</td>
<td>22.0 27.6</td>
<td>22.2 28.7</td>
<td>21.0 26.4</td>
<td>27.1 33.2</td>
<td>26.0</td>
</tr>
<tr>
<td>Typical, $p = 0.9$ [176]</td>
<td>23.7 29.2</td>
<td>24.2 30.3</td>
<td>19.4 24.5</td>
<td>24.8 32.0</td>
<td>26.0</td>
</tr>
</tbody>
</table>

**Re-ranking 20 nucleiussamples**

<table>
<thead>
<tr>
<th></th>
<th>Unigram overlap</th>
<th>LM perplexity</th>
<th>RANKGEN PG-XL-gen</th>
<th>RANKGEN PG-XL-inbook</th>
<th>RANKGEN PG-XL-both</th>
<th>RANKGEN all-XL-both</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>42.0 51.0</td>
<td>27.8 35.1</td>
<td>26.3 32.6</td>
<td>26.5 32.7</td>
<td>26.5 32.8</td>
<td>26.5 32.7</td>
</tr>
<tr>
<td></td>
<td>42.4 52.9</td>
<td>27.1 35.4</td>
<td>26.5 33.4</td>
<td>26.9 34.1</td>
<td>27.5 33.9</td>
<td>27.5 33.7</td>
</tr>
<tr>
<td></td>
<td>35.1 41.0</td>
<td>23.0 28.9</td>
<td>20.4 26.5</td>
<td>21.8 27.7</td>
<td>21.8 28.0</td>
<td>21.8 28.0</td>
</tr>
<tr>
<td></td>
<td>47.4 54.7</td>
<td>35.2 39.2</td>
<td>28.6 34.2</td>
<td>27.4 34.2</td>
<td>29.2 34.5</td>
<td>29.2 34.5</td>
</tr>
<tr>
<td></td>
<td>45.8</td>
<td>31.4</td>
<td>28.6</td>
<td>28.9</td>
<td>29.3</td>
<td>29.3</td>
</tr>
</tbody>
</table>

Table B.9: A version of Table B.8 considering only lemmatized nouns, proper nouns and numbers, with similar trends.

<table>
<thead>
<tr>
<th>Model</th>
<th>Batch</th>
<th>ChapterBreak PG19</th>
<th>AO3</th>
<th>StoryCloze 2016</th>
<th>StoryCloze 2018</th>
<th>Hella Swag 1</th>
<th>Hella Swag 3</th>
<th>Hella Swag 5</th>
<th>Hella Swag 10</th>
<th>Hella Swag 50</th>
<th>RELiC (Recall@k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>Size</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>base</td>
<td>4096</td>
<td>57.7</td>
<td>36.0</td>
<td>67.6</td>
<td>68.7</td>
<td>30.7</td>
<td>3.8</td>
<td>8.2</td>
<td>10.8</td>
<td>15.4</td>
<td>31.6</td>
</tr>
<tr>
<td>large</td>
<td>4096</td>
<td>60.6</td>
<td>31.9</td>
<td>69.3</td>
<td>69.8</td>
<td>34.2</td>
<td>5.7</td>
<td>11.0</td>
<td>14.5</td>
<td>20.0</td>
<td><strong>36.6</strong></td>
</tr>
<tr>
<td>XL</td>
<td>1536</td>
<td><strong>63.5</strong></td>
<td><strong>36.9</strong></td>
<td><strong>71.1</strong></td>
<td><strong>72.6</strong></td>
<td><strong>40.7</strong></td>
<td><strong>4.5</strong></td>
<td><strong>8.4</strong></td>
<td><strong>11.0</strong></td>
<td><strong>15.1</strong></td>
<td><strong>27.9</strong></td>
</tr>
</tbody>
</table>

Table B.10: Variation in performance on existing suffix identification and literary retrieval datasets with model size and minibatch size (number of negative samples). Overall, we see that scaling both model size and minibatch size improves suffix identification performance. See Table B.6 for comparisons with non-RANKGEN baselines.
Table B.11: Variation in performance on our PG19 / Wikipedia suffix identification datasets with model size and minibatch size (number of negative samples). Overall, we see that scaling both model size and minibatch size improves suffix identification performance. See Table 3.1 for comparisons with non-RANKGEN baselines. * Note that these numbers are lower since hard sets were adversarially constructed using this RANKGEN variant.

<table>
<thead>
<tr>
<th>Model Size</th>
<th>Batch</th>
<th>pg19-random 2-way</th>
<th>pg19-hard 11-way</th>
<th>wiki-random 2-way</th>
<th>wiki-hard 11-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>4096</td>
<td>98.6</td>
<td>91.7</td>
<td>69.4</td>
<td>36.8</td>
</tr>
<tr>
<td>large</td>
<td>4096</td>
<td>99.0</td>
<td>94.2</td>
<td>76.0</td>
<td>46.4</td>
</tr>
<tr>
<td>XL</td>
<td>1536</td>
<td><strong>99.1</strong></td>
<td><strong>94.4</strong></td>
<td><strong>78.0</strong></td>
<td><strong>49.5</strong></td>
</tr>
</tbody>
</table>

(RANKGEN models trained on PG19)

<table>
<thead>
<tr>
<th>Model Size</th>
<th>Batch</th>
<th>pg19-random 2-way</th>
<th>pg19-hard 11-way</th>
<th>wiki-random 2-way</th>
<th>wiki-hard 11-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>base</td>
<td>4096</td>
<td>97.9</td>
<td>88.4</td>
<td>63.5</td>
<td>29.8</td>
</tr>
<tr>
<td>large</td>
<td>4096</td>
<td>98.6</td>
<td>92.1</td>
<td><strong>68.6</strong></td>
<td>39.3</td>
</tr>
<tr>
<td>XL</td>
<td>256</td>
<td>96.8</td>
<td>83.7</td>
<td>60.3</td>
<td>26.0</td>
</tr>
<tr>
<td>XL</td>
<td>512</td>
<td><strong>97.7</strong></td>
<td><strong>87.8</strong></td>
<td>63.1</td>
<td>31.6</td>
</tr>
<tr>
<td>XL</td>
<td>768</td>
<td>98.1</td>
<td>89.7</td>
<td>64.7</td>
<td>34.2</td>
</tr>
<tr>
<td>XL</td>
<td>1536</td>
<td><strong>98.7</strong></td>
<td><strong>92.6</strong></td>
<td><strong>61.3</strong></td>
<td><em>39.5</em></td>
</tr>
</tbody>
</table>

(RANKGEN models trained on all 4 domains)

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Generator Language Model (re-ranking 20 nucleus samples)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>batch size  GPT2-md  GPT2-XL  T5-XXL-PG19  T5-XXL-C4  Average</td>
</tr>
</tbody>
</table>

(RANKGEN models trained on PG19 and evaluated on PG19 prefixes)

<table>
<thead>
<tr>
<th>Batch size</th>
<th>Generator Language Model (re-ranking 20 nucleus samples)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>batch size  GPT2-md  GPT2-XL  T5-XXL-PG19  T5-XXL-C4  Average</td>
</tr>
</tbody>
</table>

(RANKGEN models trained on all 4 domains and evaluated on Wikipedia prefixes)

Table B.12: Variation in MAUVE score of top-ranked generation (among 20 nucleus samples with \( p = 0.9 \)) using RANKGEN variants having a different model / minibatch size. On average, increasing model size and minibatch size boosts performance, but the trend is less prominent than in other tasks. However, all RANKGEN variants outperform baselines like nucleus sampling (see Table 3.3 for details).
<table>
<thead>
<tr>
<th>Model</th>
<th>batch size</th>
<th>GPT2-md</th>
<th>GPT2-XL</th>
<th>T5-XXL-PG19</th>
<th>T5-XXL-C4</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>(RANKGEN models trained on PG19 and evaluated on PG19 prefixes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PG19-base</td>
<td>4096</td>
<td>84.4</td>
<td>78.3</td>
<td>68.3</td>
<td>70.9</td>
<td>75.5</td>
</tr>
<tr>
<td>PG19-large</td>
<td>4096</td>
<td>93.7</td>
<td>87.9</td>
<td>79.1</td>
<td>81.3</td>
<td>85.5</td>
</tr>
<tr>
<td>PG19-XL</td>
<td>1536</td>
<td><strong>97.4</strong></td>
<td><strong>93.7</strong></td>
<td><strong>87.4</strong></td>
<td><strong>89.7</strong></td>
<td><strong>92.1</strong></td>
</tr>
<tr>
<td>(RANKGEN models trained on all 4 domains and evaluated on Wikipedia prefixes)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>all-base</td>
<td>4096</td>
<td>71.9</td>
<td>68.2</td>
<td>88.2</td>
<td>60.0</td>
<td>72.1</td>
</tr>
<tr>
<td>all-large</td>
<td>4096</td>
<td>80.4</td>
<td>74.7</td>
<td>93.0</td>
<td>64.7</td>
<td>78.2</td>
</tr>
<tr>
<td>all-XL</td>
<td>256</td>
<td>73.4</td>
<td>68.8</td>
<td>88.8</td>
<td>60.7</td>
<td>72.9</td>
</tr>
<tr>
<td>all-XL</td>
<td>512</td>
<td>78.5</td>
<td>73.6</td>
<td>93.1</td>
<td>64.3</td>
<td>77.4</td>
</tr>
<tr>
<td>all-XL</td>
<td>768</td>
<td>81.9</td>
<td>76.1</td>
<td><strong>95.4</strong></td>
<td>65.8</td>
<td>79.8</td>
</tr>
<tr>
<td>all-XL</td>
<td>1536</td>
<td><strong>84.5</strong></td>
<td><strong>78.0</strong></td>
<td>95.3</td>
<td><strong>67.3</strong></td>
<td><strong>83.7</strong></td>
</tr>
</tbody>
</table>

Table B.13: Variation in human-written text identification (vs machine generated with \( p = 0.9 \)) performance with model size and minibatch size (number of negative samples). Overall, we see that scaling both model size and minibatch size improves human text identification performance. See Table 3.2 for comparisons with causal LMs.

Figure B.6: The interface shown to Upwork freelancers for human evaluation. We used Amazon Mechanical Turk Sandbox to collect our annotations (note that we use the MTurk Sandbox interface only; we do not hire any workers from MTurk due to poor annotation quality [112].

Enter 'Text 1' or 'Text 2' followed by a 1-3 sentence explanation.

Submit
We are currently looking for people with some experience in English content writing / teaching / editing to read a prompt text (200-250 words) and choose which of two article fragments (70-100 words each) is a valid continuation of the prompt text. This study is a part of a bigger academic research project on text evaluation. If you decide to help us in this project, you will be asked to: - set up an account on Amazon Mechanical Turk Sandbox (this is what we use as the interface, payment will be through Upwork only) - read and evaluate two sets of 200 fragments, choosing which fragment is a better continuation of the prompt. You will NOT need to go through complicated and lengthy guidelines. You do NOT need to provide any written feedback on each story fragment, and you do NOT need to mark mistakes or edit the article fragments. Simply choose the fragment which continues the context better. The budget we have for this project is $100, which is calculated assuming a $25/h rate (calculated based on the average time per story fragment from the data we have already collected).

Additional instructions for adding explanations:
In this task you need to choose which better completion is better, along with 2-3 sentences explaining why you felt so. Some examples of this kind of annotation — (1) Text 1; Text 1 is more relevant to the context because .... (2) Text 2; Both texts are relevant to the context, but Text 1 has lesser repetitions and is more coherent because .... (3) Text 2; Text 2 does not contradict itself like Text 1. In general it would be great if you quote certain parts of the context / continuation to support your argument.. for instance — The context talks about the adventures of Frodo, and how he they started after "he inherited the ring from Bilbo". Text 1 goes on to talk about how Bilbo "suddenly left on his birthday" which "gave the ring to Frodo", whereas Text 2 contradicts the context by saying "Bilbo went out for the adventure with the ring."

Table B.14: The job posting and instructions shown to Upworkers before they performed the annotation task.
Table B.15: A continuation of Table 3.9 with representative model generations using RANKGen vs nucleus sampling [89], along with human explanations (from English teachers/writers) for preferring RANKGen outputs. For every row the color coding is used to ground the annotator explanation in the prefix and generation. See Table B.16-B.20 for full-length generations.
**Prefix (PG19)** A Young Inventor’s Pluck by Arthur M. Winfield: "Oh, I only wanted to know," replied the young man, apparently unconcerned. "Pooler said I was never to speak of it," returned Meg. "He thinks I don’t know where it is, but I do. " "Show it to us, will you?" asked Mont, eagerly. Meg shrank back. "It’s haunted!" she exclaimed. "Besides, Pooler would kill me if he found it out."

"We’ll never tell him, will we, Jack?" "Certainly not," replied the young machinist, to whom this new move was only another mystery. "Besides, we intend to be your friends, you know," continued Mont. "Well, then, come on," replied Meg, finally. "I don’t care if he likes it or not. 'Taint a very nice place though." "Never mind. We can’t find any worse or more dangerous places than we have already gone through." Meg picked her way along one bank of the island for nearly a quarter of a mile. Jack and Mont followed closely.

**Text 1 (RANKGEN, T5-XXL-C4):** The path led through an area of woods, shrubbery and scrub oak. Neither meadow nor open ground was seen. Jack called out as they scampered around an occasional rough rock, often from the height of a dozen feet. "I think I can see it now," said Meg, suddenly, as they rounded a grassy slope and followed another bank. A few yards farther on she stopped, and gazed at the clear sky and the undisturbed valley. "That’s the place," she said. The young man could tell she had a point.

**Text 2 (Nucleus, T5-XXL-C4):** Jack would not let her out of his sight. He made a sharp look at Meg as she walked forward. Then, his expression became grave. "Meg," he whispered, "are we going to the haunted house or not?" "This is the haunted house, Jack," she replied. "Is it haunted?" "Of course it’s haunted," returned Mont, "but if you were so scared you would be shaking all over." "Of course it’s haunted," replied the young man.

**Human Annotation 1:** Text 1. Second text appears to omit a part when they get to their destination having them appear at their destination instantly. Although a haunted house sounds more in-line with the sample text than a haunted valley, the first text fits more naturally with the sample text.

**Human Annotation 2:** Text 1 - Meg is taking them somewhere - so leading them would make sense. 'That’s the place' indicates she had found it. Jack does not sound worried in prefix text, so it would not really fit that he doesn’t let Meg out of his sight as indicated in Text 2.

**Human Annotation 3:** Text 2. The collocations and style in Text 1 sound too colloquial and off when compared to the prefix. "(...) he had a point. "

---

**Prefix (PG19)** A Young Inventor’s Pluck by Arthur M. Winfield: But at that moment, he did not stop to question how the young man had gotten into a position similar to his own. His one thought was whether or not his friend was alive. He placed the body as comfortably as possible on the fork of the tree, and then watched eagerly for some sign of life. There was a cut upon Mont’s brow, and presently a few drops of blood oozed from the wound, and trickled down his cheek. Jack took this as a good sign, and he was not mistaken, for a few seconds later the young man gave a deep sigh, and slowly opened his eyes. "Where am I?" he asked, faintly. "Here with me, Mont," replied the young machinist, bending over him. "Don’t you know me?" "Jack!" was the low response. "Oh, how my head hurts! Where are we?" "On the river." "Where are we?" Mont attempted to rise, but Jack pushed him back. "Oh, I remember now!" he continued, shuddering. "Remember what?"

**Text 1 (Nucleus, GPT2-md):** "Mont, when I was twenty-two, I met some girls who took care of my grandfather, and when I wanted to give a hand in the shop, I got him to promise to do it for me for free, if he liked me, or give me money. And now, when I tell you I feel a sense of belonging, Jack, I have been a slave for four hundred years. I have been born of slavery, and yet I feel like a slave." Jack listened to this.

**Text 2 (RANKGEN, GPT2-md):** "That time I tried to drag you off my back and then went back on to kill myself." "That is wrong," replied the young machinist, trembling. "I was in your arms. I never intended to kill myself. It happened." "It was a mistake, then," said Jack with a shake of his head. "That was just the first of many," said Mont. The young man looked at him sadly. "Don’t lie, Jack," he said gently. "You are not dead. You are alive. What’s going on?"

**Human Annotation 1:** Text 2 - Text 1 seems too mundane to be the memory on waking after being knocked out. Text 2 would fit a little better perhaps.

**Human Annotation 2:** Text 2 - Text 2 was chosen via deductive reasoning. Text 1 mentions ideas that have nothing to do with the prefix paragraph.

**Human Annotation 3:** Text 2. Text 2 answers the question that was asked at the end of the paragraph and continues with the same characters.

---

**Table B.16:** More model generations (Part 1) from each decoding method with human preferences and explanations. See our attached data submission for all 600 annotations for the 200 generation pairs.
While she was skinning the potatoes, and munching them, a noise behind her made her stop. "Janey!" she called, lifting the candle and peering into the darkness. "Janey, are you there?" A heap of ragged coats was heaved up, and the face of a young girl emerged, staring sleepily at the woman. "Deborah," she said, at last. "I'm here the night." "Yes, child. Hur's welcome," she said, quietly eating on. The girl’s face was haggard and sickly; her eyes were heavy with sleep and hunger: real Milesian eyes they were, dark, delicate blue, glooming out from black shadows with a pitiful fright. "I was alone," she said, timidly. "Where’s the father?" asked Deborah, holding out a potato, which the girl greedily seized. "He’s beyant,—wid Haley,—in the stone house." (Did you ever hear the word _jail_ from an Irish mouth?) "I came here."

I needed to find my feet.” "What have you done here?" Deborah was amazed. "Whoever came in here yesterday found his feet," she replied, "and went away—not hearing my voice." "What a wonder! What chance do you have of hearing your own voice?" She went over to the fire, lowered it, and began to put the potatoes on it, by degrees, till she had covered all the floor. "Why did you come here?" Deborah inquired, as she looked at the girl’s face. "Just for the night. Where’s the father?"

I came to ask for food. Hur git me?" "You don’t see it so bad," said Deborah, leaning her long arms about the girl’s small shoulders. "You’re poor." "That’s true," the girl said, weeping softly. "Hus poor is no better than a beast’s. I’d be well if they’d only set food on the table," she added, trembling again. "Evil grows in the dark," said Deborah sadly.

They are armed with two large claws, whose upper surfaces are split into segmented claws, to allow them to grasp with maximum force. The claws are bent backwards when not in use, to keep them from rolling off the lobster’s body, and are hooked into the shell. One claw can also bear a thick, sharp biting claw (known as the “hammer claw”). Both claws contain a fan-shaped set of “feathered” hairs, arranged around a central spine. The foot is covered in spines, and each skeletal segment has a claw (which is a pair of claws in an asymmetrical arrangement). The claws bear nails (trapezoids) and powerful radulae, and are able to claw prey through a small hole in the ground. The second pair of pereiopods has a small foot, with three claws arranged symmetrically. The foot of the second pair is covered in concave spines which are longer and thinner than those of the first pair.

Because "The first pair of pereiopods is armed with a large, asymmetrical pair of feet. They (the feet???) are armed with two large claws...) just sounds weird.

Human Annotation 1: Text 2. Both bit chaotic and funny, but the second one seems at least more outlandish. Being poor or saying Hur and Hus isn’t as bad as putting potatoes in fire that’s covering the floor.

Text 1 (Nucleus, T5-XXL-C4): They are armed with two large claws, whose upper surfaces are split into segmented claws, to allow them to grasp with maximum force. The claws are bent backwards when not in use, to keep them from rolling off the lobster’s body, and are hooked into the shell. One claw can also bear a thick, sharp biting claw (known as the “hammer claw”). Both claws contain a fan-shaped set of “feathered” hairs, arranged around a central spine.

Text 2 (RANKGEN, T5-XXL-C4): The foot is covered in spines, and each skeletal segment has a claw (which is a pair of claws in an asymmetrical arrangement). The claws bear nails (trapezoids) and powerful radulae, and are able to claw prey through a small hole in the ground. The second pair of pereiopods has a small foot, with three claws arranged symmetrically. The foot of the second pair is covered in concave spines which are longer and thinner than those of the first pair.

Human Annotation 2: Text 2 - Prefix text says she is hungry, so it would make sense that she would come to ask for food.

Human Annotation 3: Text 2. Deborah already told the girl in the prefix part where the father is, but she asks again in Text 1. So Text 2 makes more sense.

Table B.17: More model generations (Part 2) from each decoding method with human preferences and explanations. See our attached data submission for all 600 annotations for the 200 generation pairs.
Prefix (PG19) *The Horse in History* by Basil Tozer: Two years after James I. had ascended the throne there set in one of the coldest winters this country has ever known, with the result that a long stretch of the River Ouse became frozen over and so afforded the king an opportunity, of which he was quick to avail himself, of organising a race-meeting on the ice. Drake tells us that the course extended “from the tower at the end of Marygate, under the great arch of the bridge, to the crane at Skeldergate Postern.” But even so early as this in the reign of King James the opponents of horse racing began to raise indignant protests against “the folly and wickedness of betting on running horses,” protests to which but scant attention was paid. Not until some years later did the extremely zealous clergymen named Hinde set seriously to work to denounce the practice of gambling in any and every form, and he appears then to have spoken and written so forcibly that many persons of intelligence and education—I quote from a trustworthy source—gathered round and strove to encourage him to the best of their ability.

Text 1 (RANKGEN, GPT2-md): He did not, however, heed their advice and instead turned his attention to betting. “The king, who at that time was of a temperance and reform bent, had taken into his royal commission the recommendation that betting and horse-trading, while in principle harmless, be avoided. Such was the feeling among the clergy at that time as well as among the people, that in their opinions bettors and dealers, when dealing openly with men, did more harm than good.

Text 2 (Nucleus, GPT2-md): "Here again I have a record that can be said to be worthy of mention of a high level. There is one point which the facts of the situation require me to address, namely that from that day till our own day it is an object of our interest to try to remove all evidence of a race-course. That purpose is, I believe, the greatest thing that we can hope to attain in the course of this study.

Human Annotation 1: Text 1. The original paragraph mentions a "zealous clergymen" and "denounce the practice of gambling", and text 1 continues the discourse about "the clergy at the time" and "their opinions betters and dealers". Text 2 does not follow the same discourse, nor does it provide more information in prefix.

Human Annotation 2: Text 2 - prefix text is setting up for a quote with 'I quote from a trustworthy source, then Text 2 is the start of a quote as it begins with quotation marks.

Human Annotation 3: Text 1; Text 1 continues the narrative established around horse-race betting established in the prefix. The prefix mentions, "the opponents of horse racing began to raise indignant protests against [betting]", then, Text 1 continues this idea when it states, "He did not, however, heed their advice and instead turned his attention to betting".

Prefix (Wikipedia) *Thelnetham Windmill*: The mill is a small four storey tower mill with Patent sails and winded by a fantail. It drives two pairs of millstones, with a third pair driven by an auxiliary engine. The earliest record of a windmill in Thelnetham was a post mill which was shown of Joseph Hodgkinson’s map of Suffolk of 1778. In 1818, the post mill was moved to Sandy Lane, Diss, Norfolk The post mill was replaced by a tower mill, which was built by millwright George Bloomfield for William Button in 1819. The mill was set to work on Christmas Day 1819. In 1832, the mill was modernised by the fitting of a cast-iron windshaft, Patent sails and a fantail. The new windshaft was fitted on 16 July 1832. It was made by J Aickman, the King’s Lynn millwright, and weighs 1 3/4 tons (1,780 kg). A new stock was fitted in September 1836. William Button died on 11 February 1837. The mill passed jointly to his widow Rebecca and their son Richard.

Text 1 (Nucleus, GPT2-md): In 1841, it was renamed The Postmill Mill in memory of its deceased owner. The mill has been restored to its original function. It is now used to run turbines for a turbine company. The mill was built for William Button. He used to work on it during the day, but he would also travel by carriage. In the years around 1816-17, he visited several other mills in the town, including his father’s old mill. He bought it in 1810. This mill was not as modern as the mill on the left.

Text 2 (RANKGEN, GPT2-md): Richard built the mill and sold it to John Beattie on 21 May 1845. On 8 June 1854, Thomas Moore purchased the mill from the Beattie family. In 1861 the mill was owned by the Earl of Dartmouth who rebuilt it as a tenement mill for a number of years, then sold it to a local paper company for £60 a week to pay for repairs. In 1913, the mill was purchased by David Cotrell. There is a stone chimney and iron grate set into the front wall of the mill.

Human Annotation 1: Text 2. Text 2 follows the original style of the paragraph by continuing the history of the mill, while Text 1 is redundant and repeats some things from the original text. Due to following the same pattern as the original paragraph, text 2 is the best choice.

Human Annotation 2: Text 2 - I really don’t know, neither seem to fit. Text 1 says ’The mill was built for William Button’ but we’ve just been told William Button is dead so that seems to be a little odd to mention, so Text 2 is a better choice.

Human Annotation 3: Text 2; Text 2 is the better continuation of the prefix. Prefix ends with mentioned that, "The mill passed jointly to his widow Rebecca and their son Richard. Then, the Text 2 begins with, "Richard built the mill" - this connects the prefix to Text 2.

Table B.18: More model generations (Part 3) from each decoding method with human preferences and explanations. See our attached data submission for all 600 annotations for the 200 generation pairs.
Table B.19: More model generations (Part 4) from each decoding method with human preferences and explanations. See our attached data submission for all 600 annotations for the 200 generation pairs.
Prefix (PG19) Letters of Lord Acton by Lord Acton: In that character he showed, when occasion came, that his lxvi long silence in Parliament had not been due to incapacity for public speaking. At Windsor he was agreeable to the Queen from his German tastes and sympathies, not to mention the fact that he could speak German as fluently as English. Every moment of leisure during his "wait" there was spent in the Castell library. Yet the position was an unnatural one, and Lord Acton soon became anxious to escape from it. His thoughts turned to his favourite Bavaria, and he humbly suggested the Legation at Stuttgart as a possible sphere. But something infinitely better than any political or diplomatic post remained for this born student and truly learned man. In 1895, just a year after Mr. Gladstone's resignation, Sir John Seeley, Professor of Modern History at Cambridge, departed this life. The Chair was in the gift of the Crown, that is, of the Prime Minister, and Lord Rosebery appointed Lord Acton. The appointment was singularly felicitous, and the opportunity came in the nick of time.

Text 1 (RANKGEN, GPT2-md): Lord Acton entered upon his first year in the Chair, and the year was filled with extraordinary events, the first of which was the assassination of King George V, which had so disastrous an effect upon the world that it has never since been fully restored. Then came the attempt on the life of Sir Edmund Burke at St. Petersburg, the downfall of William of Orange, and the beginning of the Spanish War.

Text 2 (Nucleus, GPT2-md): The "youngest and most handsome man of his class", as they called him, had been nominated to succeed him. Mr. Gladstone, whose wit had been so well known for many years, was now at last able to be seen in public with dignity, and his manner was becoming more relaxed. The Prince felt confident enough to nominate Sir John, who was the most distinguished scholar in Bavaria, and he received his nomination in person. There were some curious features of the office.

Human Annotation 1: Text 1 - The prefix paragraph focuses on the life and career of Lord Acton, thus, Text 1 is the best selection to continue exposition of Lord Acton's career.

Human Annotation 2: Text 1 - Prefix states that Lord Acton was appointed as chair, Text 1 then mentions Lord Acton entering his first year as chair. This seems to follow more closely than Text 2.

Human Annotation 3: Text 3: At the end of the paragraph, Lord Acton is appointed Chair and text 3 delves into what occurred once that was fact. Text 2 does not closely follow the narrative of the paragraph.

Prefix (Wikipedia, Battle of Roanoke Island): They arrived too late to take part in the battle, but were involved in the attack on the fortress at Pao-ting Fu, where the Chinese government was believed to have found asylum after Peking was captured by western forces. The Victorians joined a force of 7,500 men on a ten-day march to the fort, once again only to find that it had already surrendered. The Victorians then garrisoned Tientsin and the New South Wales contingent undertook garrison duties in Peking. HMCS Protector was mostly used for survey, transport, and courier duties in the Gulf of Chihli, before departing in November. The naval brigades remained during the winter, unhappily performing policing and guard duties, as well as working as railwaymen and fire-fighters. They left China in March 1901, having played only a minor role in a few offensives and punitive expeditions and in the restoration of civil order. Six Australians died from sickness and injury, but none were killed as a result of enemy action.

Text 1 (Nucleus, GPT2-md): In July 1901 they began their work in the Indian Ocean, arriving in Hong Kong in November 1901. In January 1902, the Australian Expeditionary Force joined the Chinese in a massive invasion of Peking. While Australian naval forces took an active part in this event, the Australian Navy was still engaged in military operations in the Gulf of Hainan. When the Chinese invaded Hong Kong, the navy withdrew from the mainland and its warships were deployed at the port of Hong Kong until December 1902.

Text 2 (RANKGEN, GPT2-md): A second invasion of the Chinese mainland was attempted by British naval forces on 23 June 1901 when Victoria signed a treaty with China. Following the treaty signing, a force led by HMCS Sault Ste. Marie was sent to occupy Peking and Tientsin. HMAS San Francisco, HMAS Mackellar and HMAS Melbourne returned to Hong Kong after a two-month deployment in China in early 1902 and were replaced by a group of 14,000 men under HMCS Leutl, which was formed on 24 November 1902 as part of the second invasion.

Human Annotation 1: Text 2; Text 2 is the better continuation of the prefix. In Text 1, it isn’t clear who "they" is in the phrase, "they began their work in the Indian Ocean' which makes Text 1 appear disjointed when reading directly after the prefix whereas Text 2’s introduction flows more seamlessly even though it’s introduction brings a slight change in idea.

Human Annotation 2: Text 1. Although both texts could follow the paragraph, Text 1 follows along with the timeline set in the paragraph.

Human Annotation 3: Text 2 - very difficult without more knowledge of these events. I’m picking text 2 just because the date mentioned, 23 June 1901, is closest to the date mentioned in prefix text - march 1901

Table B.20: More model generations (Part 5) from each decoding method with human preferences and explanations. See our attached data submission for all 600 annotations for the 200 generation pairs.
C.1 Bootstrap analysis of inter-annotator variance

We utilize the bootstrap resampling [254] technique described in [39] to estimate confidence intervals for human evaluation data. At a high level, bootstrap resampling helps capture the uncertainty in a downstream test statistic by repeatedly sampling from the data with replacement. We consider two downstream test statistics in our work — (1) average system level performance; (2) correlation of human judgements to automatic metrics.

While [39] were primarily interested in uncertainty due to the specific instances and systems evaluated, our goal is to capture uncertainty due to the inter-annotator variance. Hence unlike [39], we sample with replacement from the set of annotators for every instance. Our precise formulation can be found in Algorithm 1, which operates on a $X \in \mathbb{R}^{N \times M}$ matrix of human annotations where $N$ is the number of summaries, and $M$ the number of annotators.

**Algorithm 1 Bootstrap Confidence Interval**

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>samples ← an empty list</td>
</tr>
<tr>
<td>2</td>
<td>for $k$ iterations do</td>
</tr>
<tr>
<td>3</td>
<td>$X_s$ ← empty $N \times M$ matrix</td>
</tr>
<tr>
<td>4</td>
<td>for $i \in {1, \ldots, N}$ do</td>
</tr>
<tr>
<td>5</td>
<td>$D \leftarrow$ samp. ${1, \ldots, M}$ w/ repl. $M$ times</td>
</tr>
<tr>
<td>6</td>
<td>for $j \in {1, \ldots, M}$ do</td>
</tr>
<tr>
<td>7</td>
<td>$X_s[i, j] \leftarrow X[i, D[j]]$</td>
</tr>
<tr>
<td>8</td>
<td>end for</td>
</tr>
<tr>
<td>9</td>
<td>end for</td>
</tr>
<tr>
<td>10</td>
<td>Calculate test statistic on $X_s$ and append to samples</td>
</tr>
<tr>
<td>11</td>
<td>end for</td>
</tr>
<tr>
<td>12</td>
<td>$\ell, u \leftarrow (\alpha/2) \times 100$ and $(1 - \alpha/2) \times 100$ percentiles of samples</td>
</tr>
<tr>
<td>13</td>
<td>return $\ell, u$</td>
</tr>
</tbody>
</table>
Table C.1: Fleiss kappa (F-κ), Randolph kappa (R-κ), and agreement scores of our FINE annotation per summary unit. All κ scores are well above a random annotation baseline, indicating good agreement.

<table>
<thead>
<tr>
<th></th>
<th>F-κ</th>
<th>R-κ</th>
<th>all agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.00</td>
<td>0.00</td>
<td>25%</td>
</tr>
<tr>
<td>SQuALITY</td>
<td>0.74</td>
<td>0.76</td>
<td>82%</td>
</tr>
<tr>
<td>PubMed</td>
<td>0.53</td>
<td>0.65</td>
<td>74%</td>
</tr>
</tbody>
</table>

C.2 Human evaluation details

C.2.1 FINE-grained evaluations of SQuALITY and PubMed summaries

We interviewed a total of 9 Upwork freelancers for the position, offering a compensation of $15-16.5 / hr (depending on their Upwork hourly rate). The screening procedure involved a qualification task on synthetically perturbed summaries from the SQuALITY dataset validation split. Similar to the final annotation task, annotators were shown a highlighted clause from the summary, and asked to mark whether or not it is supported by the source document. 50% of the clauses were synthetically perturbed (via negation or entity swapping as in [137]) and manually checked to ensure they were not supported by the source document. A total of 6 freelancers scored 85% or better, and were recruited for the main set of experiments. All 9 freelancers were compensated for the screening round at the rate of 15$ USD / hr.

All six hired annotators are native or bilingual English speakers. All annotators have completed a degree at the undergraduate level and three also have Masters degrees, with the most common focuses of the degrees being English/creative writing and education. The annotators’ common professional experiences include copywriting, editing, proofreading, writing, and teaching. Finally, for PubMed annotations we re-hired three annotators from the pool of six SQuALITY annotators who mentioned they had experience reading and analyzing biomedical articles. These three annotators were provided with an additional bonus of $30 after they completed all annotations.

Annotators are provided with a detailed annotation guideline along with examples of faithfulness (Table C.4). Our guidelines are mostly consistent with a recently proposed set of guidelines for checking attribution in text generation [217]. The final annotation interface is implemented in AMT Sandbox, as shown in Figure C.4.

180
Inter-annotator agreement (binary): Much of the analysis in Section 4.3 uses standard deviation across summaries scores to measure inter-annotator agreement. However, another way to calculate inter-annotator agreement for FINE annotations is measuring agreement on individual units which received a Yes / No judgment. In Table C.1 we show these inter-annotator agreement statistics. We measure Fleiss Kappa [61], Randolph Kappa [215, 281], and the fraction of sentence pairs with total agreement. In the table we can see all agreement statistics are well away from a uniform random annotation baseline, indicating good agreement.

C.2.2 COARSE-grained evaluation of PubMed summaries

None of the surveyed papers evaluating PubMed summaries with humans released their human evaluation data. Hence, we decided to collect our own COARSE annotations. Since FINE annotations (Section C.2.1) may have biased our original set of annotators, we hire three new annotators to perform overall assessments on a 5-point Likert scale. In other words, we use a “between-subject” experiment design to compare FINE against COARSE.

We hired three freelancers on Upwork, all of whom have extensive professional experience reading research papers (two of them had PhDs in biomedical fields). All annotators were compensated at a rate of 20$ USD / hr, their hourly rate on Upwork. All three annotators had been previously screened and hired by us for different projects in the past. Two of them had assisted us in an annotation task involved reading short summaries of biomedical academic papers and evaluating them for fluency, accuracy, correctness.

Annotators are provided with a detailed annotation guideline along with examples of faithfulness (Table C.5). Our guidelines are mostly consistent with a recently proposed set of guidelines for checking attribution in text generation [217]. The final annotation interface is implemented in LabelStudio, as shown in Figure C.5.

C.2.3 Crowdworkers or expert annotators?

Several prior works have raised the issue of low inter-annotator agreement and poor accuracy with non-expert annotators (eg: MTurk crowdworkers) in human evaluation of summarization [66, 55, 56] and open-ended long-form generation [112, 32]. In our survey (Table C.3), we found the type of annotators used in long-form summarization is often not specified (16 / 43 papers). Among other papers, 10 papers use non-experts while 17 papers use expert annotators (often graduate students).

The $\kappa$ scores are measured using the library https://github.com/statsmodels/statsmodels.
Overall, we echo the concerns with non-expert annotators and recommend hiring freelancers on Upwork (or experts) who are well-versed with the domain for annotation. In initial experiments, we attempted to recruit Amazon Mechanical Turk crowdworkers filtered by the “Master’s qualification” and having a 90%+ approval rating. In our qualification task of error detection in synthetically perturbed SQuALITY summaries, MTurkers scored just 62% (binary classification) with a three-annotator Fleiss $\kappa$ of 0.15. On the other hand, Upwork freelancers (with professional writing experience) an accuracy 90% with a high inter-annotator agreement (Fleiss $\kappa = 0.71$).

### C.3 Additional Survey Statistics

In Table C.2 and Table C.3 we document some additional statistics for the 44 papers conducting human evaluation of long-form summarization.

<table>
<thead>
<tr>
<th>Best practice</th>
<th># papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw human evaluation data released</td>
<td>2 / 44</td>
</tr>
<tr>
<td>Interface or instructions provided</td>
<td>9 / 44</td>
</tr>
<tr>
<td>Inter-annotator agreement reported</td>
<td>12 / 44</td>
</tr>
<tr>
<td>Statistical analysis conducted</td>
<td>12 / 44</td>
</tr>
<tr>
<td>Multiple datasets are human evaluated</td>
<td>14 / 44</td>
</tr>
<tr>
<td>Multiple annotators per summary</td>
<td>33 / 44</td>
</tr>
<tr>
<td>Annotator background reported</td>
<td>33 / 44</td>
</tr>
<tr>
<td>Specific summary aspects evaluated</td>
<td>42 / 44</td>
</tr>
</tbody>
</table>

Table C.2: Fraction of surveyed papers following the best practices recommended by [64]. We include only the 44 papers here which conducted a human evaluation of long-form summarization.

<table>
<thead>
<tr>
<th>Type of annotator</th>
<th># papers</th>
</tr>
</thead>
<tbody>
<tr>
<td>No details specified</td>
<td>11 / 44</td>
</tr>
<tr>
<td>Native English speaker**</td>
<td>5 / 44</td>
</tr>
<tr>
<td>Mechanical Turk crowdworker</td>
<td>9 / 44</td>
</tr>
<tr>
<td>Non-expert volunteers</td>
<td>1 / 44</td>
</tr>
<tr>
<td>Extensive prior experience**</td>
<td>3 / 44</td>
</tr>
<tr>
<td>Graduate students / researchers</td>
<td>13 / 44</td>
</tr>
<tr>
<td>Upwork freelancers</td>
<td>2 / 44</td>
</tr>
</tbody>
</table>

Table C.3: The types of annotators used across different long-form summarization papers. ** - No additional details were specified.
C.4 Automatic summarization metrics used for evaluation

The following metrics are considered while measuring Pearson’s correlation with our human evaluation data (Figure 4.2) — ROUGE-1 / 2 / F [161], BARTScore / BARTScore-Parabank [305], Sentence-BLEU [197], BERTScore [313] and BLEURT [227]. A number of metrics were calculated using the SacreROUGE repository [40].

C.5 Learning effect while annotating long-form summaries

In Section 4.3.3 we discussed a learning effect where annotators get more familiar with the contents of a source document as they annotate more fine-grained units in a long-form summary. To better understand this effect, in Figure C.1 we plot the average time taken by annotators as they progress in their annotation of a summary. Overall, we find that annotators get significantly faster in annotating the summary after the first 20% units. We hypothesize that annotators get pretty familiar with the general topics in the source document after the first few annotations, speeding up subsequent annotations.

Figure C.1: Learning effect over time while evaluating long-form summaries with FINE annotation. As the annotators evaluate more summary units, they learn the document better and are much faster at annotation irrespective of whether hints are shown to them.

C.6 Partial summary annotation with Pearson correlation

See Figure C.2.

C.7 Metric correlations using Kendall’s Tau

See Figure C.3.
Figure C.2: A version of Figure 4.4 using Pearson correlation instead of Kendall Tau correlation.

Figure C.3: A version of Figure 4.2 using Kendall’s Tau correlation instead of Pearson’s correlation.
Figure C.4: The AMT Sandbox annotation interface used for **FINE** evaluation of SQuALITY and PubMed summaries (Appendix C.2.1).

Figure C.5: The LabelStudio annotation interface used for **COARSE** evaluation of PubMed summaries (Appendix C.2.2).
In this task, you will be shown a long document ("Source Document") and its Summary. A span of text will be highlighted in the summary, and the goal is to check if this span is factually supported by the source document. You will need to choose one of two options:

1. **Yes**: if all the facts in the highlighted summary span are supported by the source document
2. **No**: if the highlighted summary span presents some information that is not supported by the source document (either a direct contradiction, or not present)

In addition to the source document, you will be provided with some highlighted text ("hints") in the source document which may help you in making a decision. Press the "Next Hint" button to scroll through the highlighted hints. Source document hints may or may not be helpful. Do not make a judgment solely based on these hints. Skim through the source document yourself / search for keywords with Ctrl + F if the hints are not helpful.

Below you can find some short representative examples.

**Example 1**
Summary (only highlighted span shown) = ... Retief is not Lemuel’s cousin. ...
Source Document (snippets shown) = He eyed Retief ... "He ain’t no cousin of mine," Lemuel said slowly.
Supports = Yes

**Example 2**
Summary (only highlighted span shown) = ... Lemeul knocks down Retief. ...
Source Document (snippets shown) = Retief’s left fist shot out, smacked Lemuel’s face dead center. He stumbled back, blood starting from his nose; ... He caught himself, jumped for Retief ... and met a straight right that snapped him onto his back: out cold. "Wow!" said Potter. "The stranger took Lem ... in two punches!"
Supports = No (Reason: Retief knocks down Lemeul, not the other way around.)

**Example 3**
Summary (only highlighted span shown) = ... Potter and his team do not trust the Embassy. ...
Source Document (snippets shown) = Lemme up. My name’s Potter. Sorry ’bout that. I figured it was a Flap-jack boat; looks just like ’em . He waved a hand toward the north, where the desert lay.
Supports = No (Reason: The claim is irrelevant to the evidence.)

Table C.4: Annotation guidelines provided to annotators for FINE-grained evaluation of SQuALITY and PubMed summaries. (Appendix C.2.1).
Instructions for Likert-scale evaluation. Please read all instructions before starting the annotation.

Setup
1. Start by signing up on Label Studio, you will need to provide an email ID and password. It’s okay to use a non-existent throw-away email ID here. Also, do not use any personal / sensitive passwords (but make sure to remember your email / password for logging in next time!). Click on the box saying “<your name> — Summarization Evaluation”
2. In this batch a total of 30 summaries need to be evaluated. Every three consecutive rows are different summaries of the same source document. You can evaluate a summary by clicking on a row, and annotating it. Optionally, you can click on “Label All Tasks” at the top of the screen.

Annotation Task
Each summary needs to be evaluated for its “correctness”. You need to provide a 0-5 judgment for the entire summary, where “correctness” can be defined as, “The absence of factual errors in the summary, where a factual error is a statement that contradicts the source document, or is not directly stated, heavily implied, or logically entailed by the source document”. For example, Source Document (snippet shown) = . . . Vitamin C was discovered in 1912, isolated in 1928, and, in 1933, was the first vitamin to be chemically produced. It is on the World Health Organization’s List of Essential Medicines. Vitamin C is available as an inexpensive generic and over-the-counter medication. Partly for its discovery, Albert Szent-Györgyi and Walter Norman Haworth were awarded the 1937 Nobel Prizes in Physiology and Medicine and Chemistry, respectively. Foods containing vitamin C include citrus fruits, kiwifruit, guava, broccoli, Brussels sprouts, bell peppers, potatoes, and strawberries. Prolonged storage or cooking may reduce vitamin C content in foods. . . . .
Summary 1 (snippet shown) = . . . Chicken contains vitamin C . . .
Summary 2 (snippet shown) = . . . Albert Szent-Györgyi won the 1955 Nobel Prize for discovering Vitamin C . . .
Summary 3 (snippet shown) = . . . Vitamin C was the first chemically produced Vitamin . . .
Summary 4 (snippet shown) = . . . Apple contains vitamin C . . .
Errors marked in red. Here, the snippets for summary 1 are incorrect, summary 2 partially correct, and summary 3 completely correct with respect to the source document. Summary 4 is incorrect with respect to the source document (since it’s never discussed), but a globally correct fact. You should treat such a summary as incorrect since it is not mentioned in the source document.
(This is an illustrative example only, the actual annotation task has much longer summaries / source documents.)
The rating scale is from 0 to 5, where 0 is the lowest possible rating (most or all of the summary is wrong / irrelevant to the source document), and 5 is the highest rating (most or all of the summary is correct).
While it is compulsory to provide a judgment from 0 to 5 for each summary, you can optionally provide additional comments in your annotation. For instance, if the judgment needs to be more nuanced than a 5-point scale, you prefer to mark something like “3.5”, or you would like to add some other notes about your judgment.
Press “Submit” after you have provided your annotation.

Suggested workflow
Every three consecutive rows contain different summaries for the same source document. We suggest the following workflow while annotating documents —
1. Spend the first 15 minutes reading the source document and getting a general sense of the facts mentioned in the document.
2. Spend 5 minutes to read and annotate the summaries in each of the three consecutive rows which correspond to the same document. Add optional comments / notes if necessary.
3. In the last 5 minutes, re-calibrate your ratings across the three rows if needed (for instance, you significantly preferred the correctness of summary 1 vs summary 2, but you gave it the same rating in the initial pass). Add optional comments / notes if necessary.
Following this workflow, it should take 35 minutes to annotate each set of 3 rows. For 30 rows, this should take 6 hrs.

Table C.5: Annotation guidelines provided to annotators for COARSE evaluation of PubMed summaries (Appendix C.2.2).
APPENDIX D

APPENDIX FOR “AI-GENERATED TEXT DETECTION: PARAPHRASING ATTACKS AND RETRIEVAL DEFENSES”

D.1 Ethical Considerations

Our goal in this chapter is not to provide a recipe for potential attackers (e.g., college students wishing to use ChatGPT in their essays) to evade AI text detection systems. Rather, we wish to bring awareness to the wider community about the vulnerabilities of current AI-generated text detectors to simple paraphrase attacks. These detectors are not useful in their current state given how easy they are to evade. We encourage the research community to stress test their detectors against paraphrases, and to develop new detectors which are robust against these attacks. To facilitate such research, we open source our paraphraser and associated data / code.

Furthermore, we propose not just an attack but also a potentially strong defense against this attack. Our detection strategy is simple, relying on retrieval over a corpus of previously-generated sequences. We empirically show that such a detection algorithm could work at scale and provide extensive discussion on possible methods to improve performance (Appendix 5.5.4), as well as discussing possible limitations and approaches to tackling them (Appendix 5.5.5). We hope that retrieval-based AI-generated text detectors rapidly improve and are eventually deployed in conjunction with other detection methods like watermarking / classifiers.

D.2 Related work for discourse paraphrasing

D.2.1 Survey of paraphrase generation papers

As an important NLP task, paraphrasing has attracted much attention. Many models have been proposed to improve the quality of paraphrases. To position our model DIPPER and highlight its strengths, we conduct a survey of paraphrase generation papers from 2018 to 2022 (Table D.1) and focus on the following four aspects:

1. Whether a model can paraphrase a paragraph at once,
2. whether a model can merge or split consecutive sentences when appropriate,
3. whether a model leverages context surrounding an input sentence when paraphrasing,
4. whether a model provides control knobs for users to customize the output diversity.
The survey shows that only three out of 25 papers mentioned that their model can paraphrase more than one sentence (but not necessarily at once). None of them enables their model to merge or split sentences when paraphrasing. No model uses information from context surrounding an input sentence during inference time. Finally, 14 papers offer ways for users to customize the diversity of paraphrases. However, most diversity control methods such as constituency parses or exemplars may not be straightforward and intuitive to end-users as the scalar control knobs in DIPPER.

In contrast to the papers in the survey, DIPPER nicely combines all desiderata into one model and offers intuitive control knobs for lexical and syntactic diversity. Automatic and human evaluation show that DIPPER can efficiently leverage context information and reorganize sentences while having high fidelity in meaning (Appendix 5.6).

D.2.2 Other related work

In this section we discuss a few additional less related papers which were not included in our survey in Appendix D.2.1. Our discourse paraphraser is closely related to work on contextual machine translation, where source/target context is used to improve sentence-level machine translation [93, 105, 277, 255, 139, 4, 180, 310, 298, 104, 265, 60, 170]. Prior work has shown that context helps with anaphora resolution [267], deixis, ellipsis, and lexical cohesion [266]. Efforts to make paraphrase generation more contextual have been quite limited. A few efforts have attempted to use sentence-level context to paraphrase phrases [34, 174], and dialogue context to paraphrase individual dialogues in a chat [63].

Our work is also related to efforts in text simplification to go beyond a sentence, by collecting relevant datasets [301, 41] and building unsupervised algorithms [143]. Note that our work focuses on a general-purpose paraphrasing algorithm and is not tied to any particular style, but could be utilized for document-level style transfer using techniques like [134, 130]. Similar efforts have also been undertaken in machine translation, [204, 110, 171], attempting to translate paragraphs/documents at once.

D.3 More experimental details of our attack experiments

D.3.1 Details for training our paraphraser DIPPER

Our paraphraser DIPPER is a sequence-to-sequence Transformer neural network [262], initialized with the T5-XXL 1.1 checkpoint [211] and fine-tuned on our paraphrase generation data, using early stopping on validation loss for held-out novels. We find it helpful to paraphrase a maximum of 3 consecutive sentences at time, which leads to better adherence to control codes. Our models are implemented in JAX [22] using the T5X library [218] with the default fine-tuning hyperparameters. Training was done on 32 cloud TPUv3 chips, and took 6-12 hours to complete. At inference time, we use nucleus sampling [89] with $p = 0.75$ and a variety of control codes.

To make our chapter more intuitive, we have slightly modified the notation that our actual pretrained model uses. Our pretrained model uses control codes $100 - L$ and $100 - O$, denoting lexical/order similarity rather than diversity. Also, $\langle sent \rangle$ is used instead of $\langle p \rangle$. We clearly document this in the code release.
<table>
<thead>
<tr>
<th>Paper</th>
<th>Multi-sentence</th>
<th>Merge / Splits</th>
<th>Contextual</th>
<th>Diversity Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>[101]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Constituency parse</td>
</tr>
<tr>
<td>[159]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[221]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[292]</td>
<td>✓</td>
<td>?</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[141]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[95]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Decoding constraints</td>
</tr>
<tr>
<td>[29]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Exemplar</td>
</tr>
<tr>
<td>[160]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Granularity control¹</td>
</tr>
<tr>
<td>[70]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Exemplar</td>
</tr>
<tr>
<td>[153]</td>
<td>✓</td>
<td>?</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[250]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>n-gram overlap</td>
</tr>
<tr>
<td>[140]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Exemplar</td>
</tr>
<tr>
<td>[116]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[134]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[212]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[177]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Diversity score³</td>
</tr>
<tr>
<td>[96]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Constituency parse</td>
</tr>
<tr>
<td>[162]</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[69]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Binary</td>
</tr>
<tr>
<td>[35]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
<tr>
<td>[45]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>n-gram</td>
</tr>
<tr>
<td>[13]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Control code⁴</td>
</tr>
<tr>
<td>[92]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Syntactic sketch</td>
</tr>
<tr>
<td>[303]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td>Exemplar+Keywords</td>
</tr>
<tr>
<td>[297]</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
<td></td>
</tr>
</tbody>
</table>

**DIPPER (ours)** ✓ ✓ ✓ ✓ ✓

Table D.1: The table shows the result of our survey of paraphrase generation papers from 2018 to 2022. We focus on four aspects: (1) whether a model can paraphrase multiple sentences at once, (2) whether a model is able to merge or split an input sentence when appropriate, (3) whether a model takes context surrounding the input sentence into consideration when paraphrasing, and (4) whether a model enables users to control the semantic and syntactic diversity of paraphrases. ¹Granularity levels are *word*, *phrase*, and *sentence*. ²[177] use context for their dataset construction, but do not leverage it during training/inference. ³The diversity score is a combination of the unigram Jaccard distance and the relative position change for unigrams. ⁴The code is represented by a three dimensional vector corresponding to semantic similarity as well as syntactic and lexical distances between the input and output sentences.
D.3.2 Long-form question answering data processing

In Section 5.4 evaluate long-form question answering [57], in which an LM must answer a how/why question (e.g., Why are almost all boats painted white?) with a 250-350 word answer. To build a long-form question answering dataset, we scrape questions from the r/explainlikeimfive subreddit posted between July to December 2021. We randomly sample 500 questions from each of six popular domains on the subreddit (biology, physics, chemistry, economics, law, and technology) and pair each question with its longest human-written answer, which yields 3K long-form QA pairs.

D.4 Controlled comparisons of retrieval with other AI-generated text detectors on open-ended text generation

We conduct a controlled comparisons of retrieval on the open-ended text generation task with Wikipedia prompts (see Section 5.5.2). The result of the experiment is presented in Table D.2.

<table>
<thead>
<tr>
<th>Open-ended text generation with Wikipedia prompts (300 generated tokens)</th>
<th>GPT2-XL</th>
<th>OPT-13B</th>
<th>GPT-3.5 (davinci-003)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>+ 60L</td>
<td>+ 60L,60O</td>
<td>Original</td>
</tr>
<tr>
<td><strong>Baseline methods:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watermark</td>
<td>100.0</td>
<td>68.9</td>
<td>57.2</td>
</tr>
<tr>
<td>DetectGPT</td>
<td>70.3</td>
<td>8.7</td>
<td>4.6</td>
</tr>
<tr>
<td>OpenAI</td>
<td>21.6</td>
<td>13.3</td>
<td>14.8</td>
</tr>
<tr>
<td><strong>(Ours) Retrieval over corpus of 3K generations from model itself, with retriever:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>100.0</td>
<td>86.4</td>
<td>81.5</td>
</tr>
<tr>
<td>BM25</td>
<td>100.0</td>
<td>99.0</td>
<td>98.0</td>
</tr>
<tr>
<td><strong>(Ours) Retrieval over corpus of 9K generations pooled from all three models, with retriever:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SP</td>
<td>100.0</td>
<td>72.1</td>
<td>63.2</td>
</tr>
<tr>
<td>BM25</td>
<td>100.0</td>
<td>85.0</td>
<td>78.7</td>
</tr>
</tbody>
</table>

Table D.2: Our retrieval defense significantly improves AI-generated text detection accuracy (at 1% FPR) over baselines on all settings, including our most diverse paraphrase attacks (+60L and +60L,60O).

D.5 ROC curves at different FPR

See Figure D.1.

---

1We choose this period since current language models have been trained on internet data available before June 2021 [193], this prevents verbatim copying from training data.
Figure D.1: ROC curves for text generated by GPT2-XL, before paraphrasing (solid lines) and after paraphrasing (dashed lines, pp). Different plots represent different clipping thresholds on the X-axis.