Understanding the Limitations of Using Large Language Models for Text Generation

Daphne Ippolito

A DISSERTATION

in

Computer and Information Science

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2021

Supervisor of Dissertation

Chris Callison-Burch, Professor, Computer and Information Science

Graduate Group Chairperson

Mayur Naik, Professor of Computer and Information Science

Dissertation Committee

Lyle Ungar, Professor, Computer and Information Science

Dan Roth, Professor, Computer and Information Science

Marianna Apidianaki, Professor, Computer and Information Science

David Grangier, Research Scientist, Google Brain

Understanding the Limitations of Using

Large Language Models for Text Generation

ACKNOWLEDGEMENT

Understanding the Limitations of Using
Large Language Models for Text Generation

ABSTRACT

State-of-the-art neural language models are capable of generating incredibly fluent English text. In my proposed dissertation, I aim to explore how these models can be applied for creative purposes, such as story generation and building creative tools for writers. I also address several challenges around evaluating the use of neural language models for open-ended text generation. I propose an analysis of the tradeoff between generating lexically diverse text and text humans perceive as high quality. I introduce a detection-based evaluation task that can be used to investigate this tradeoff as well as to study quality differences between natural language generation systems. Lastly, I propose an investigation into the extent to which the giant language models used for text generation produce text which verbatim copies from the training set.

CONTENTS

1 INTRODUCTION

One of the oldest yet most elusive promises of AI is computers that can converse with humans, not just via rigidly structured templates and programming languages, but in natural language. This problem is formalized as Natural Language Generation (NLG), the task of writing novel sentences in a human language such as English. Modelling developments over the last half-decade have led to natural language generation systems which are capable of producing incredibly fluent text. These systems have been applied to practical domains such as machine translation and text summarization and simplification, but they have also been applied to more fanciful ones, such as story generation, video games development, and tooling for creating writing.

Story generation and creative writing are a particularly interesting domain to investigate the strengths and weaknesses of natural language generation systems. Unlike in machine translation or summarization, where it is critical that the generated text is factual and stays faithful to the source material, in creative domains, the hallucinations and unusual word choices that are pervasive in modern NLG may be beneficial to a creative writer's process. Indeed, ideation tools, tools which suggest writing topics, are commonplace in creative writing circles. Furthermore, AI-assisted creative writing is a useful testbed for the strengths and limitations of modern NLG systems, one that challenges how controllable these systems are and whether they can be interacted with in useful ways.

Chapter **??** describes my contributions to the field of AI-assisted creative writing. It discusses the importance of introducing controllability into natural language generation

systems–providing writers the ability to dictate what kind of text gets generated and decide how it interfaces with what they might have already written. In particular, we introduce methods for efficiently supporting a fill-in-the-blank paradigm, where a writer can insert text into any position of their current text **{TODO: citation}**. We also describe a simple recipe for supporting style-transfer into any user-defined style without the need for costly training data acquisition and test-specific model training. Both these approaches are incorporated into Wordcraft, an AI-augmented text processor that provides several interfaces for writers to get feedback and suggestions from an NLG system. In studies with both amateur and skilled writers, we found Wordcraft to be a valuable assistive tool for creative writing.

Modern natural language generation systems rely on neural language models, neural networks trained on billions of words of text in order to represent human language (Chapter 2). Understanding the limitations of these language models and the nature of the text they are capable of generating is crucial to the ultimate use of these systems in real applications such as Wordcraft.

A significant limitation for creative applications like Wordcraft is the difficulty in generating text that is diverse (containing uncommon and interesting words and phrases) and high quality (as perceived by human readers). Chapter **??** focuses on this challenge and its ramifications to the detectability of generated text. While many approaches have been proposed to, given a language model, try to improve the diversity of its generations without sacrificing quality, we show that none quite achieve this goal [18]. Most often, practitioners choose approaches that err on the side of human-perceived quality at the expense of lexical diversity. This decision leaves subtle signatures in the generated text which make it easy for automatic classifiers to distinguish it from genuine human-written text.

The remainder of Chapter **??** focuses on this detection problem. The proliferation of machine-generated text, especially when it lacks attribution, is of significant concern to the public. As tools for generating text become more powerful and easier to use, they could enable nefarious uses such as fabricated product reviews and social media posts. In Ippolito et al. 2019 [17], we measure the ability of humans as well as automatic systems to detect machine-generated text. Understanding detectability is imperative because it gives us a proxy for how far along generative systems are at fooling humans and whether undesired use of machine-generated text can be mitigated.

One difficulty in studying human ability to detect machine-generated text is that it can be very difficult to collect annotations. Many of the errors NLG systems make are subtle and require closely reading several sentences of text to be able to identify. Common strategies for soliciting human annotators, such as paying crowd workers a fixed dollar amount per annotation, do not tend to yield useful annotations since annotators are not incentivized to spend the extra time to do a close read. In order to be able to study human detection ability at scale, we built the Real or Fake Text game (RoFT), a website that gamifies the task of identifying machine-generated text [11]. The RoFT platform allowed us to collect over 40,000 annotations of whether players could correctly identify when a passage of text transitioned from being human-written to being machine-generated . Chapter **??** presents a detailed analysis of the factors we found that most impacted detectability.

Of course, machine-generated text is most undetectable when it looks *exactly* like its training data. Large languages are worryingly capable of memorizing and regurgitating significant amounts of their training data. For example, GPT-3, a popular model that has already been incorporated into several products, when prompted with the first sentence of *Harry Potter* or *Lord of the Rings* will accurately generate the first several paragraphs of each book, This behaviour is especially problematic for the domain of AI-assisted

creative writing, as writers using tools such as Wordcraft have the expectation that the generations they are being shown are unique and not plagiarized. Memorization also makes the task of studying the dectability of generated text more challenging. If an NLG system generates Chapter 1 of *Harry Potter*, should this text be labeled as human-written or machine-generated? Chapter **??** focuses on this question of memorization. First, we show how performing thorough deduplication of training data results in models that are less likely to exhibit memorization . Then we conduct experiments showing how observable memorization scales with respect to the number of times a sequence occurs in the training set, the model size, and the length of conditioning prompt [5].

1.1 SUMMARY OF CONTRIBUTIONS

something something something

2 BACKGROUND ON TEXT GENERATION

Automatic text generation has been a goal of computer science researchers since the early days of computing. In recent years, algorithm and statistical approaches have given way to neural language models–neural networks trained to build representations of human language from millions or even billions of documents. This chapter gives a brief overview of how the task of text generation was approached before the ascendency of deep learning and then delves into the details of modern text generation using neural language models.

2.1 BRIEF HISTORY OF TEXT GENERATION

2.2 WHAT IS A LANGUAGE MODEL?

A language model is any model that assigns probabilities to sequences of words. Given a sequence of words $w_1, \ldots, w_n$, a langueg model outputs the likelihood $P(w_1, \ldots, w_n)$ of this sequence. An ideal language model would have high likelihood to natural-sounding text, like the sentences in this paragraph, and low likelihood to gibberish. Most language models the assumption that the likelihood of a word is dependent only on the words that precede it. Thus, the chain rule applies:

$$P(w_1, \ldots, w_n) = P(w_1) \times \ldots \times P(w_i|w_1, \ldots, w_{i-1}) \times \ldots \times P(w_n|w_1, \ldots, w_{n-1}) \tag{2.1}$$

Before the transition to neural network-based models, the most common form of language model was an n-gram model. Instead of trying to estimate the probability of a word given all preceding words, n-gram model make the Markov assumption that the probability of a word is only dependent on a fixed number of preceeding words, referred to as an n-grams. For example, using a 2-gram model, we would approximate each factor in Equation 2.1 as

$$P(w_i|w_1, \ldots, w_{i-1}) \approx P(w_i|w_{i-2}, w_{i-1}) \tag{2.2}$$

An n-gram models can be constructed from a corpus of text by simply counting how many times each word in the text is preceeded by each possible n-gram.

There are several disadvantages to this n-gram based approach to language modelling. First, n-gram models tend to be sparse. If a particular <n-gram, word> pair never occurs in the corpus, then the model will assign it a probability of 0. As a result, smoothing techniques are often employed to prevent plausible but novel word sequences from being assigned a probability of zero. Second, the complexity of storing an n-gram language model grows exponentially with the choice of n. In practice, most n-gram models used n between 1 and 5, which is insufficient for modelling long-term coherence. Third, n-gram models cannot represent words which are not in their vocabulary. Such words are typically replaced with a special out-of-vocabulary identifier. Neural language models, described in the following sections, overcome many of these limitations.

2.3 WHAT IS A NEURAL LANGUAGE MODEL?

Neural network-based language models replace the statistical models described in the previous section with a learned function (the neural network) whose output can be used to predict the likelihood of a word sequence. In contrast to n-gram models, neural language models are capable of assigning non-zero probability to sequences never seen in their training corpora, and thus they can be used to model sequences of hundreds or even thousands of words.

One of the key advancements in neural language modeling was the transition from operating on sequences of discrete words to operating on sequences continuous vector representations. The sequence of words $w_1, \ldots, w_n$ is mapped to sequence of embedding vectors $\mathbf{y}_1, \ldots, \mathbf{y}_n$. In early work on neural language modeling, these vector representations were computed separately. Algorithms such as word2vec [33] and GloVe [36] were employed to consturct embedding matrices where each row corresponded to a word in the chosen vocabulary. In today's neural language models, the embedding matrix is typically treated as part of the neural language model, initialized randomly than optimized along with the reset of the network. Let $\mathbf{E}_\theta$ be a learned embedding matrix where each row correspond to the vector represention of one word in the vocabulary.

Typical neural language models emit $\hat{\mathbf{y}}_t$, a predicted embedding for the tth position in the sequence given the previous word embeddings in the sequence. This can be written as

$$\hat{\mathbf{y}}_t = f_\theta(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}) \tag{2.3}$$

where f_θ is the neural network and $\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}$ are the embeddings of the previous tokens in the sequence.

To produce a probability distribution for what the next word should be given the previous words, the predicted embedding $\hat{\mathbf{y}}_t$ is multiplied by the embedding matrix $\mathbf{E}_\theta$ to produce a score for each word in the vocabulary. Then a softmax transformation is used to normalize these scores into a probability distribution. Let Y_t be a random variable representing the vocabulary item predicted for the tth position. We then have:

$$P(Y_t = i|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])} \tag{2.4}$$

where i and j are indexes into the vocabulary.

The learned weights θ are optimized using a log likelihood loss. More precisely, we can write the training loss for a sequence $\mathbf{y}_1, \ldots, \mathbf{y}_n$ as:

$$\mathcal{L} = -\sum_{t=1}^{n} \log P(Y_t = i^*|\mathbf{y}_{1:t-1}) \tag{2.5}$$

$$= -\sum_{t=1}^{n} \log \frac{\exp(\mathbf{E}_\theta\hat{\mathbf{y}}_t[i^*])}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j])} \tag{2.6}$$

$$= -\sum_{t=1}^{n} \mathbf{E}_\theta\hat{\mathbf{y}}_t[i^*] \tag{2.7}$$

$$= -\sum_{t=1}^{n} (\mathbf{E}_\theta f_\theta(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1})[i^*] \tag{2.8}$$

In these equations, i^* is the index of the groundtruth word at position t in the sequence. By taking the dot product between the neural network's predicted embedding and the embedding of the true word at each position t (Eq. 2.7), we get a score for how correct the neural network's prediction for this position is. Training with an objective of maximizing the sum of these scores over every word position is equivalent to minimizing the negative log likelihood (or maximizing the likelihood) of the sequence.

In some language modelling application, it is common to have an additional sequence which the model is conditioned on in addition to the tokens of the target sequence. This paradigm is known as an encoder-decoder or sequence-to-sequence, and the formulation above is modified to

$$\hat{\mathbf{y}}_t = f_\theta(\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}; \mathbf{x}_1, \ldots, \mathbf{x}_n) \tag{2.9}$$

where $\mathbf{x}_1, \ldots, \mathbf{x}_n$ is the additional input sequence. The most popular application of encoder-decoder models is machine translation, where to convert some text from French to English, the language model predicts the next word of the English sequence given the entirety of the French sequence and the preceedings words of the English sequence.

Most state-of-the-art neural language models uses a variant of the Transformer architecture [44] as the neural network f_θ. All of the research presented in this dissertation uses the Transformer architecture, except for Chapter 3.3, which uses an older LSTM-based recurrent architecture [16].

2.4 ENCODING TEXT INTO A VOCABULARY

For simplicity, the previous sections refer to the input to a language model as a sequence of words, but in practice, neural language models use a variety of different techniques to construct vocabularies of varying granularities. There is no single solution for forming the base units of language (referred to for the remainder of this chapter as "tokens"), and techniques vary significantly across languages. In English, the simplest vocabularies are character-level–each letter of the alphabet and punctuation becomes a token. Historically, word-level vocabularies, where each token corresponds to a word in the dictionary, were

most common. Word-level vocabularies can be created by splitting a string on whitespace and punctuation. Since the space of possible words is near-boundless, in practice only the most common tens or hundreds of thousands of words are included in the vocabulary, and all other words are replaced with an out-of-vocabulary token.

In recent years, subword vocabularies have become standard in neural language modeling. Subword vocabularies are formed by choosing a budget (the desired size of the vocabulary), then running an algorithm that joins letters together into larger units, such that the most common character sequences end up as tokens in the vocabulary. While common words such as “cat” or “dog” end up as single tokens in the vocabulary, uncommon words such as hippopotamus end up bring broken into multiple tokens. Several greedy algorithms have been proposed to approximate optimally breaking up a text corpus into subwords, but byte-pair encoding (BPE) is currently the most popular [40]. Typical subword vocabulary sizes are between 32,000 and 50,000 tokens.

Table 2.1 shows the same string under a few different tokenization schemes.

Table 2.1: Examples of the string “A hippopotamus ate my homework." tokenized using three different vocabularies. With the subword tokenizer, the rare word “hippopotamus" gets broken up into multiple tokens. For word-level tokenizers, if the word “hippopotamus” occurred very infrequently in the corpus used to build the vocabulary (or perhaps the writer of the sentenced misspelled it), it would typically get replaced with an out-of-vocabulary token (row 4).

Vocab Type	Example
character-level	`['A', ' ', 'h', 'i', 'p', 'p', 'o', 'p', 'o', 't', 'a', 'm', 'u', 's',' ', 'a', 't', 'e', ' ', 'm', 'y', ' ', 'h', 'o', 'm', 'e', 'w', 'o', 'r', 'k', '.']`
subword-level	`['A', 'hip', '##pop', '##ota', '##mus', 'ate', 'my', 'homework', '.']`
word-level	`['A', 'hippopotamus', 'ate', 'my', 'homework', '.']`
word-level	`['A', '[UNK]', 'ate', 'my', 'homework', '.']`

For all of the types of vocabularies discussed, a decision must be made on whether to convert strings to lowercase before vocabulary creation. Removing case allows for a more compact vocabulary, but it also removes potentially useful information about the location of proper nouns.

Subword vocabularies were designed to be a compromise between the advantage and disadvantages of word-level and character-level vocabularies. Character-level vocabularies are usually very small, no more than a couple hundred tokens. However, the vocabulary can cover near every possible string a person could write. Word-level vocabularies cannot feasible contain the hundreds-of-thousands of words present in English text. Realistically, only the most common words are kept, and less common ones are replaced with a special `UNK` token. When text is tokenized with character-level vocabularies, the resulting sequences are very long, while word-level tokenization yields shorter sequences since there is just one token per word. Lastly, word-level representations learned by a neural net tend to more meaningful than character-level representations since a word has semantics associated with it that are common across uses. Subword vocabularies adopt the best of both worlds, using word-level tokens for common words but falling back to subword, or in the worst case, character-level, tokenization for uncommon words. This approach eliminates the need for an out-of-vocabulary token and results in tokenized sequence lengths which are somewhere between the two strategies.

2.5 GENERATING TEXT WITH A LANGUAGE MODEL

Neural language models in themselves are not generative. As described in the previous sections, most language models provide a probability distribution for what the next token in the sequence *could* be, given the previous tokens. To perform generation, an algorithm

for decoding sequences from these predicted probability distributions is needed. At each step of decoding, the decoding algorithm performs a forward pass on the neural network using the existing prompt text as input, selects a next token based on the neural network's predictions, adds this token to the prompt, and repeats until the desired number of tokens have been generated.

There are many possible strategies for making a token selection at each step. The simplest strategy is to take the arg max of the distribution, greedily picking the token with the highest probability according to the model. This approach is simple but only allows a single generation to be produced for any given prompt. Alternatively, one can randomly sample from the vocabulary, where each vocab item is assigned a sampling probability proportional to its probability predicted by the language model. This method allows for many different sequences to be generated from the same prompt. However, in practice, this tactic results in noisy, low-quality text, as the probability distributions returned by neural language models tend to be very long tailed, and the chance of sampling a word from this long tail is quite high.

Several strategies have been proposed to improve random sampling techniques by reducing the entropy of the distribution before sampling. Introducing a temperature parameter τ into the softmax computation allows us to smoothly shift probability mass from low-scoring item in the vocabulary to high-scoring ones.

$$P(Y_t = i|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}) = \frac{\exp(\mathbf{E}\hat{\mathbf{y}}_t[i]/\tau)}{\sum_j \exp(\mathbf{E}\hat{\mathbf{y}}_t[j]/\tau)} \tag{2.10}$$

Alternatively, one can introduce sparsity intro the distribution by deliberately zeroing out like-likelihood vocabulary items. Top-k random sampling accomplishes this by restricting sampling to only the k most likely tokens at each step. Nucleus sampling, also referred to as top-p random sampling, accomplishes this by restricting sampling at timestep t to the

k_t most likely tokens, where k_t is selected such that these tokens cover no more than p% of the probability mass. For all three of these techniques there is a parameter (τ, k, or p) which controls the amount of randomness we want to permit in the generation. Choosing a low value for these parameters results in an increasingly peaky distribution, which, at its extreme, is the same as taking the arg max. Choosing a high value for these parameters results in the distribution that looks closer and closer to the original scores produced by the model. The diversity-quality tradeoff that results from choice of these parameters is described in detail in Chapter **??**.

Lastly, search methods such as beam search can be used to try and find the most likely overall sequence. Actually computing the most likely sequence is intractable due to the exponential search space, but algorithms such as beam search so a good job of approximating it. In practice, these techniques tend to lead to text that is bland and repetitive, Chapter 3.3 goes into detail about why this is the case.

2.6 CONTROLLABILITY AND TASK-SPECIFIC GENERATION

In the early days of neural language modelling, it was common to train a separate neural language model for each generative task of interest. For example, if one wanted a system capable of producing chatbot dialog, one would train their neural language model on a dialog dataset such as OpenSubtitles [46]. If one wanted a system able to perform text summarization, one would likewise train a model from scratch on a dataset such at Gigaword [35]. At the time, the neural networks being used for these sorts of tasks were relatively small, and training and maintaining one model per task, was mostly feasible.

In 2018, Radford et al. [37] proposed the idea of training a single "General Purpose Transformer (GPT)" model which could be subsequently finetuned for any number of

language tasks. The idea of finetuning a more general model for a specific task had already taken off in computer vision, where researchers had shown a convolutional neural network trained on the ImageNet task of classifying the contents of images could be finetuned for tasks ranging from image segmentation to **{TODO: }**.

General purpose language models intended for generative tasks tend to be trained on massive datasets scraped from the internet, such as C4 [39] or The Pile [12]. It is common to use both decoder-only models trained only to predict the next token given the previous ones [38], as well as encoder-decoder architectures trained with a de-noising loss, where the input is a corrupted version of the text, and the task is to recover the uncorrupted text [39, 25]. Finetuning such models has yielded immense success in tasks across the field of natural language processing. Chapter **??** focuses on the feasibility of finetuning for the fill-in-the-blank task.

There are however several limitations to the paradigm of pre-training followed by fine-tuning. As state-of-the-art neural language models increase in number of parameters, the computational expense of finetuning is becoming increasingly prohibitive. Furthermore, the need to store (potentially in GPU memory) one set of model weights per task makes it very difficult to build downstream applications which need to perform several different tasks. In addition, finetuning only works where there is enough data to fine-tune one. Overfitting is a significant challenge in low-resource settings, where there may only be a handful of training examples.

For these reasons, various approach have been proposed for replacing the finetuning step with methods which require either no or minimal weight training. Brown et al. [4] introduce the technique of few-shot prompting. By constructing a textual prompt which contains several examplars of the goal task, a general-purpose language model can be made to perform the task. Chapter **??** uses this technique for the task of textual style, while Chapter **??** shows how it can be used for a variety of story editing operations. Lester

et al. [24] introduce *prompt tuning* as an improvement over few-shot prompting that trains a small neural network to produce an optimal prompt in embedding-space for the goal task.

3 FACTORS THAT IMPACT GENERATED TEXT QUALITY, DIVERSITY, AND DETECTABILITY

3.1 MOTIVATION

It is not a question that natural language generation systems have made leaps and bounds in the last decade. But how do we evaluate this progress and determine whether one NLG system on average produces better outputs than another NLG system? In my research, I show how "better" is difficult to define, and I propose the task of detection–identifying whether a piece of text was written by a human or generated by a system–as a way to better understand and compare the strengths of different NLG systems.

It turns out that choosing the choice of decoding strategy (see Section **{TODO: }**) has a huge impact on the quality of the text that gets generated with a neural language model. Search-based decoding strategies which optimize for choosing the mostly likely overall sequence end up producing text which is much less diverse than text a human writer would produce. This may be tolerable for applications where diversity isn't strictly needed, such as machine translation, but it is problematic for applications where there are many valid ways to continue a prompt. In this chapter, I present investigations of how the choice of decoding strategy creates a trade-off between the human-assessed quality of generated text and the amount of lexical diversity present in said text.

In Section 3.3, we consider search-based decoding strategies, focusing especially on diversity-promoting modifications of beam search. We compared several methods which

had previously never been compared with each other, and we showed that none of these diversity-promoting methods improve diversity without serious cost to generation quality. No method outperformed the others on both diversity and quality.

In Section **??**, we focus on probabalistic generation methods which randomly sample a next word using the distribution predicted by the language model. The simplest strategy is to sample directly from the predicted distribution. However, when many low-likelihood words cumulatively contain quite a bit of probability mass, choosing one of these words can lead to odd or contradictory phrases and semantic errors. Humans readers are quick to notice these types of errors. Thus, more commonly, methods are used that reduce the entropy of the distribution before sampling, which improves generation quality at the cost of diversity. We show that humans have a hard type identifying that text is machine-generated when sampling is heavily restricted to only high-likelihood words.

Finally in Section **??**, we conduct a large-scale study of the detectability of generated text by human annoators, expanding upon the small-scale human evaluation experiments described in Section **??**.

3.2 TERMINOLOGY

3.2.1 Diversity

The term “diversity” has been used in the language model literature to refer to a diverse set of properties. Some use it as a synonym for sentence interestingness or unlikeliness [15]. Others consider diversity a measure of how different two or more sentences are from each other [45, 13]. In some framings, diversity is measured across a set of generations coming from the same prompt. Given a particular prompt or input, the goal is to measure

the breadth of possible generations the model will produce [32]. Diversity can also be measured as a corpus-level: given all the sentences generated by the model for all prompts, what is the overall lexical diversity **{TODO: citation}**?

In Section 3.3, we define diversity as the ability of a generative method to create a set of possible outputs that are each valid given a particular input but vary as widely as possible in terms of word choice, topic, and meaning. We also use human evaluation to measure generation interestingness. In Section **??**, we consider decoder-only language models, where no additional input is conditioned on. In this setting, we instead consider corpus-level diversity across all the model's generations. In both sections, diversity of a set of model generations is automatically measured using distinct-n, the number of unique n-grams within the set divided by the total number of n-grams.

3.2.2 Quality

"Quality" is also a difficult property to define. When measured in downstream applications, it can be quantified as how many times a user interactions with the generative system (for example, the number of conversation turns with a dialog system) before losing interest. It can also be evaluated directly by asking a human rater, "how good is this text?", though definitions of "good" vary widely across the literature **{TODO: add citation}**.

To some extent, quality can also be measured automatically. In tasks with a clear goal, like machine translation or summarization, one can compare the generation against a gold standard. Generally, quality is strongly associated with fluency, and in **{TODO: citation}** we show that up until a point, the lower perplexity text is assigned by a language model, the more fluent it is.

Sections 3.3 and **??** take quite different approaches to measuring generation quality. In 3.3, we consider a dialog model, and ask humans to assess the quality of model

responses on three axes: fluency, adequacy, and interestingness. In **??**, we propose a novel method for assessing generation quality based on the premise that humans (or a trained discriminator) ought to have a hard time distinguishing between real human-written text and model outputs when the model outputs text that is high-quality.

3.3 DIVERSITY-PROMOTING SEARCH-BASED DECODING METHODS

3.3.1 Introduction

In 2018, state-of-the-art neural language models were based on recurrent neural networks, such as LSTMs, that struggled to generate sequences that were both long and high-quality. However, these models were beginning to have huge success over statistical techniques in natural language processing tasks such as machine translation [41, 31], text summarization [34], and dialog systems [**Vinyals2015ANC**].

At the time, the standard convention for generation was to try to generate the most likely overall sequence from the language model. This approach made a lot of sense in machine translation, where generating one correct translation was more important than generating diverse translation. Since computing the overall most likely output sequence is intractable, early work in neural machine translation found that beam search was an effective strategy to heuristically sample sufficiently likely sequences from these probabilistic models [41]. However, as neural language models came to be applied increasingly to open-ended tasks, beam search was found to be ill-suited to generating a set of diverse candidate sequences; this is because candidates outputted from a large-scale beam search often only differ by punctuation and minor morphological variations [27].

There are a number of reasons why it is desirable to produce a set of diverse candidate outputs for a given input. For example, in collaborative story generation, the system makes suggestions to a user for what they should write next [8]. In these settings, it would be beneficial to show the user multiple different ways to continue their story. In image captioning, any one sentence-long caption is probably missing some information about the image. Krause et al. [20] show how a set of diverse sentence-length image captions can be transformed into an entire paragraph about the image. Lastly, in applications that involve reranking candidate sequences, the reranking algorithms are more effective when the input sequences are diverse. Reranking diverse candidates has been shown to improve results in both open dialog and machine translation [26, 27, 13]. Furthermore, in open-ended dialog, the use of reranking to personalize a model's responses for each user is a promising research direction [7].

With these sorts of applications in mind, a variety of alternatives and extensions to beam search were proposed which sought to produce a set of diverse candidate responses instead of a single high likelihood one [26, 45, 22, 42]. Many of these approaches show marked improvement in diversity over standard beam search across a variety of generative tasks. However, as of 2018, there had been little attempt to compare and evaluate these strategies against each other on any single task.

In this sub-chapter, we survey methods for promoting diversity during decoding in order to systematically investigate the relationship between diversity and perceived quality of output sequences. We focus on conditional language models–models that are trained to map from some input, perhaps an image or some text, to a target sequence. In addition to standard beam search and greedy random sampling, we compare several recently proposed modifications to both methods. We present a detailed comparison of existing diverse decoding strategies on two tasks: open-ended dialog and image captioning, and recommendations for a diverse decoding strategy.

3.3.2 Diverse Decoding Strategies

There are many ways to decode text from a conditional language model. Let $\mathbf{y}$ represent the sequence of tokens for the target sequence ($\mathbf{y} = y_1 \dots y_n$) and $\mathbf{x}$ represent the input (which may be another token sequence, or, in the case of image captioning, an image). Most decoding strategies strive to find the most likely overall sequence, i.e. pick a $\hat{\mathbf{y}}$ such that:[1]

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} P(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} \prod_{t=1}^{N} P(y_t \mid y_{<t}, \mathbf{x})$$

In practice, this can't be computed exactly, no sub-exponential algorithm exists for find the optimal decoded sequence, and thus the field instead use approximations.

Beam search approximates finding the most likely sequence by performing breadth-first search over a restricted search space. At every step of decoding, the method keeps track of b partial hypotheses. The next set of partial hypotheses are chosen by expanding every path from the existing set of b hypotheses, and then choosing the b with the highest scores. If the goal is to approximate finding the most likely over-all sequence, the log-likelihood of the partial sequence is used as the scoring function. Algorithm 1 gives an overview of the beam search algorithm.

Since beam search only explores a limited portion of the overall search space, it tends to yield multiple variants of the same high-likelihood sequence, sequences that often only differ in punctuation and minor morphological changes [27]. Therefore, standard beam search is not ideal for producing diverse outputs. In this section, we will discuss a variety of methods that have been developed recently to eliminate redundancy during decoding and generate a wider range of candidate outputs.

1 The formulation we are using here is slightly different than Section {**TODO:** }, in that we use $\mathbf{y}$ and $\hat{\mathbf{y}}$ to refer to sequence of tokens, rather than framing in terms of a sequence of embedding vectors.

Algorithm 1 Beam Search Inference

```
1:  procedure Beam Search
2:      B ← {SOS}
3:      k ← BeamWidth
4:      out ← k-best output list
5:      while |out| < k do
6:          front ← remove all nodes fromB
7:          for w ∈ front do
8:              succ ← w's k-best successors
9:              for s ∈ succ do
10:                 if s == EOS then
11:                     out ← out ∪ {s}
12:                 else
13:                     B ← B ∪ {s}
14:                 end if
15:             end for
16:         end for
17:         Sort B
18:         if |B| > k then
19:             Prune B to k-best successors
20:         end if
21:     end while
         return out
22: end procedure
```

NOISY PARALLEL APPROXIMATE DECODING Introduced by Cho [6], NPAD is a technique than can be applied to any decoding setting. The main idea is that diversity can be achieved more naturally by taking advantage of the continuous manifold on which neural nets embed language. Instead of encouraging diversity by manipulating the probabilities outputted from the model, diverse outputs are instead produced by adding small amounts of noise to the hidden state of the decoder at each step. The noise is randomly sampled from a normal distribution. The variance is gradually annealed from a starting σ_0 to 0 as decoding progresses (that is $\sigma_t = \frac{\sigma_0}{t}$) under the reasoning that uncertainty is greatest at the beginning of decoding. NPAD can be used in conjunction with any decoding strategy; following the best results from the original paper, we show results using NPAD with beam search.

Extensions to NPAD have sought to learn the direction in which to manipulate the hidden states using an arbitrary decoding objective [14]. Since such objectives can be highly domain-specific, we do not evaluate this method.

TOP-g CAPPING In beam search, it is often the case that one hypothesis h is assigned a much higher probability than all other hypotheses, causing all hypotheses in the next step to have h as their parent. Following Li and Jurafsky [27] and Li et al. [28], we add an additional constraint to standard beam search to encourage the model to choose options from diverse candidates. At each step t, current hypotheses are grouped according to the parental hypothesis they come from. After grouping candidates, only the top g from each grouping are considered. The resulting $b \times g$ candidates are ranked, and the top b are selected as hypotheses for the next beam step.

HAMMING DIVERSITY REWARD Vijayakumar et al. [45] proposes adding an additional diversity-promoting term, θ, to the log-likelihood before reranking. This term measures

how different a candidate hypothesis $c_{\leq t}^{(i)}$ is from the partial hypotheses selected in the previous step. Let $\mathcal{H}_{t-1} = \{c_{\leq t-1}^{(1)}, \ldots c_{\leq t-1}^{(b)}\}$ be these partial hypotheses. Then the beam search scoring function for the ith candidate at timestep t becomes:

$$\text{score}(c_{\leq t}^{(i)}) = \sum_{j=1}^{t} \left(\log P(c_j^{(i)} | c_{<j}^{(i)}, \mathbf{x}) \right) + \lambda \theta(c_{\leq t}^{(i)}, \mathcal{H}_{t-1})$$

where λ is a tunable hyperparameter. Vijayakumar et al. [45] try a variety of definitions for θ, including embedding diversity and n-gram diversity, but they find that Hamming distance, the number of tokens in the candidate sequence which exist in the previously selected partial hypotheses, is most effective. We take the negative of the Hamming distance as θ.

ITERATIVE BEAM SEARCH In an attempt to improve the size of the search space explored without sacrificing runtime, Kulikov et al. [22] propose an iterative beam search method. Beam search is run many times, where the states explored by subsequent beam searches are restricted based on the intermediate states explored by previous iterations. Formally, we can define the set of all partial hypotheses for beam search instance i at time step t as $\mathcal{H}_t^{(i)}$. From here, the search space explored by beam search instance i can be expressed as $S_i = \cup_{t=1}^{T} \mathcal{H}_t^{(i)}$. The ith beam search is prevented from generating any partial hypothesis that has previously been generated, that is, any hypothesis found in $S_{<i} = \cup_{i'=0}^{i-1} S_{i'}$.

The authors also attempt a soft inclusion criterion, where any states within ϵ Hamming distance from a previously explored state are also excluded. During the experimentation of Kulikov et al. [22], however, the soft-inclusion was found to not be beneficial; thus, we only restrict exact matches of previous states in our implementation. In practice, this means after the first beam search instance runs as normal, the first step of the second

beam search instance will contain the b+1 to 2b-most likely starting tokens; this pattern holds for the third beam search instance, and so on.

CLUSTERED BEAM SEARCH Most recently, Tam et al. [42] proposed a clustering-based beam search method to help condense and remove meaningless responses from chatbots. Specifically, at each decoding step t, this method initially considers the top $2 * b$ candidates. From there, each candidate sequence is embedded[2], and the embeddings are clustered into c clusters using K-means. Finally, we take the top $\frac{b}{c}$ candidates from each cluster. Note that in the case any clusters have size less than $\frac{b}{c}$, we then include the highest-ranked candidates not found after clustering.

CLUSTERING POST-DECODING (PDC) All the previous methods modified the decoding algorithm to encourage diversity. However, it is also possible to encourage additional diversity post-hoc by sampling several generations and then choosing the most diverse one. On the task of sentence simplification, after decoding using a large-scale diversity-promoting beam search (beam size 100), Kriz et al. [21] then clustered similar sentences together to further increase the variety of simplifications from which to choose. Document embeddings generated via Paragraph Vector [23] were used as the sentence embeddings with which to perform K-means.

In this work, we extend this post-decoding clustering idea in three key ways. First, we make use of sentence-level embeddings which leverage the pre-trained language representations from the Bidirectional Encoder Representations from Transformers (BERT) [10].[3] Second, after clustering, Kriz et al. [21] took the sentence closest to the centroid of each cluster as the representative candidate; we instead choose the highest-ranked

2 We follow Tam et al. [42] and used averaged GloVe word embeddings [36].
3 BERT sentence-level embeddings were obtained using https://github.com/hanxiao/bert-as-service.

candidate (according to log-likelihood) from each cluster to ensure the best candidates are still selected. Finally, after performing standard K-means clustering, we found that it was often the case that some clusters contained large numbers of good candidates, while others contained very few candidates that are also either ungrammatical or otherwise inferior. Thus, in our implementation, we remove clusters containing two or fewer sentences, and then sample a second candidate from each of the remaining clusters, prioritizing selecting candidates from larger clusters first.

3.3.3 Experimental Setup

We evaluate the decoding strategies described in the previous section under the following settings. For each of the published beam search algorithms, we choose the hyperparameters that were found to be best in the original publications.

Method	Setting
RS	Random sampling with temp = 0.5, 0.7, 1.0, or 1.0 with top-10 capping.
Standard BS	Standard beam search
Top5Cap BS	Top-g capping with $g = 3$
Iter5 BS	Iterative beam search with 5 iterations
HamDiv0.8 BS	Hamming Diversity with $\lambda = 0.8$
Cluster5 BS	Clustered beam search with 5 clusters
NPAD0.3 BS	Noisy Decoding with $\sigma_0 = 0.3$

For random sampling, we sample 10 outputs, and with beam-search based methods, we use a beam size of 10 to generate 10 outputs. In addition, we show results from oversampling then filtering. We use a beam size of 100 or generate 100 samples through

random sampling, and then we select 10 from the 100, either through post-decoding clustering (PDC) or by taking the 10 candidates with highest likelihood.

We examine these decoding strategies on two tasks: open ended dialog and image captioning. For each task, we evaluate both the quality and diversity of the 10 outputs from each strategy.

Open-ended Dialog Task

In the dialog domain, we use an LSTM-based sequence-to-sequence (Seq2Seq) model implemented in the OpenNMT framework [19]. We match the model architecture and training data of Baheti et al. [3]. The Seq2Seq model has four layers each in the encoder and decoder, with hidden size 1000, and was trained on a cleaned version of OpenSubtitles [43] to predict the next utterance given the previous one.

Evaluation is performed on 100 prompts from the Cornell Movie Dialog Corpus [9]. These prompts are a subset of the 1000 prompts used in Baheti et al. [3], which were filtered using item response theory for discriminative power.

We report perplexity (PpL), averaged over *all* the top 10 outputs for each example.[4] Since the quality of open-ended dialog is notoriously difficult to evaluate automatically, we ran a human evaluation task on Amazon Mechanical Turk where annotators were shown a prompt and 5 potential responses generated by any of our decoding methods. Evaluators were asked to provide binary ratings on fluency, adequacy, and interestingness for each response. Overall, we collected 3 human judgments for each of the top ten responses for each of our decoding methods; in other words, we collected 3,000 judgments per method. Figure **??** shows the instructions given to the human raters.

4 This differs from existing work which computes perplexity over only the top output for each example. For our task we are interested in the quality of all of the generated responses.

Please Note

- You have to be an **English Native Speaker**.
- You have to complete the judgments for all sentences. **All fields are required.**

Instructions

In this task you will read part of a conversation. You will see a series of prompts, and possible responses to these prompts created by a computer program. The program attempts to generate responses that are relevant, while also making an interesting contribution to the conversation.

For each prompt, you will read five possible responses, and make the following judgments:

1. Grammatical: Whether or not the response is a correct English sentence.
2. Coherent: Whether or not the response is appropriate given the prompt.
3. Interesting: Whether or not the response makes an interesting and informative contribution to the conversation.

Example

Below we show a prompt and several possible responses, along with a suggested score for each response.

Prompt: What do you do for work?

Responses	Grammatical?	Coherent?	Interesting?
What the what?	No	No	No
I'm a what?	Yes	No	No
I like pancakes.	Yes	No	Yes
How are you?	Yes	No	No
I don't know.	Yes	Yes	No
What?	Yes	Yes	No
I'm an artist.	Yes	Yes	Yes
I work at the police department.	Yes	Yes	Yes

Prompts and Responses

1. **Prompt:** i told you the facts! he abandoned us -- those are the facts.

   System 1: i'm sorry i'm not a criminal
   Grammatical Response? Yes No Coherent Response? Yes No Interesting Response? Yes No

   System 2: i'm not sure that's what you're gonna do
   Grammatical Response? Yes No Coherent Response? Yes No Interesting Response? Yes No

   System 3: we can not use -- against us
   Grammatical Response? Yes No Coherent Response? Yes No Interesting Response? Yes No

   System 4: what have you done?
   Grammatical Response? Yes No Coherent Response? Yes No Interesting Response? Yes No

   System 5: i observe your question sir
   Grammatical Response? Yes No Coherent Response? Yes No Interesting Response? Yes No

Figure 3.1: The full instructions for our Amazon Mechanical Turk task to evaluate the quality of our dialog system responses.

Method		Fluency	Adequacy	Interestingness	Ppl	Dist-1	Dist-2	Ent-2	Ent-4
Reference		0.795	0.732	0.636	–	–	–	–	–
RS 0.7	(sample 10)	**0.758**	0.399	**0.388**	35.98	0.63	0.80	4.08	3.84
RS 1.0	(sample10)	0.550	0.303	0.386†	67.99	**0.74**	**0.87**	**4.35**	**4.08**
RS 1.0,top10	(sample 10)	0.745†	**0.418**	0.387†	**10.33**	0.60	0.80	4.12	3.91
Standard BS	(10 beams)	**0.950**	**0.621**	0.336	**4.01**	0.37	0.45	3.16	3.01
Top3Cap BS	(10 beams)	0.942†	0.603	0.346	4.03	0.37	0.46	3.17	3.03
Iter5 BS	(10 beams)	0.903	0.520	0.335	5.42	**0.62**	**0.74**	**3.68**	**3.25**
HamDiv0.8 BS	(10 beams)	0.923	0.599	0.366†	4.56	0.33	0.37	3.08	3.00
Cluster5 BS	(10 beams)	0.936	0.582	**0.381**	4.23	0.39	0.46	3.24	3.06
NPAD0.3 BS	(10 beams)	0.942†	0.604†	0.335	4.05	0.36	0.44	3.13	2.99
RS 1.0,top10	(sample 100, rank)	**0.922**	**0.548**	0.347	**5.10**	0.52	0.68	3.54	3.18
RS 1.0,top10	(sample 100, PDC)	0.852	0.494	**0.372**	6.96	**0.63**	**0.76**	**3.74**	**3.27**
Standard BS	(100 beams, rank)	**0.964**	**0.611**	0.332†	**4.01**	0.44	0.61	3.33	3.05
Standard BS	(100 beams, PDC)	0.944	0.599	**0.346**	4.42	**0.57**	**0.70**	**3.59**	**3.21**

Table 3.1: Results on 100 dialog prompts. The first row shows the mean human ratings of the single reference response available for each prompt. The next three rows show results for random sampling, with 10 samples drawn per prompt. The next six rows are variants of beam search using beam size 10. The last four rows use random sampling or standard beam search to generate 100 outputs, then filter down to 10 outputs either through ranking by log-likelihood or by performing post-decoding clustering (PDC). In each section, the highest value is bolded, and statistical ties are marked †.

Image Captioning Task

For image captioning, we use a state-of-the-art model introduced in Anderson et al. [2]. We take advantage of Luo [30]'s open-source implementation and released model parameters trained on MSCOCO [29]. We evaluate on a test set containing 5000 images.

We report Semantic Propositional Image Caption Evaluation (SPICE) scores, an automatic evaluation metric that has been shown to correlate well with human judgments of quality[1]. SPICE measures how well the semantic scene graph induced by the proposed caption matches one induced by the ground truth. In addition to computing SPICE on the top-scoring caption (SPICE@1), we follow Vijayakumar et al. [45] in reporting Oracle SPICE@10 scores. This is done to show the upper bound on the potential impact diversity can have. We also compute the mean SPICE score across all of the candidate captions for an image. Unlike SPICE@1 and SPICE@10, this metric shows the overall quality of *all* of

Method		SPICE Mean	SPICE @1	SPICE @10	Dist-1	Dist-2	Ent-2	Ent-4
RS 0.7	(sample10)	**0.170**	**0.192**	**0.278**	0.31	0.52	3.67	4.00
RS 1.0	(sample10)	0.133	0.167	0.247	**0.44**	**0.71**	**4.17**	**4.26**
RS 1.0,top10	(sample10)	0.159	0.183	0.272	0.33	0.59	3.90	4.17
Standard BS	(10 beams)	0.194	0.193	0.283	0.18	0.26	2.94	3.18
Top3Cap BS	(10 beams)	**0.195**	**0.196**	0.282	0.17	0.26	2.93	3.17
HamDiv0.8 BS	(10 beams)	0.194	0.194	0.282	0.18	0.27	2.98	3.19
Cluster5 BS	(10 beams)	0.191	0.194	**0.285**	**0.19**	**0.28**	**3.04**	**3.25**
NPAD0.3 BS	(10 beams)	0.191	0.192	0.280	0.18	0.26	2.94	3.17
RS 1.0,top10	(sample100, rank)	**0.182**	**0.188**	**0.284**	0.25	0.41	3.31	3.64
RS 1.0,top10	(sample100, PDC)	0.169	**0.188**	0.282	**0.31**	**0.52**	**3.62**	**3.91**
Standard BS	(100 beams, rank)	**0.188**	0.190	0.279	0.20	0.31	3.04	3.32
Standard BS	(100 beams, PDC)	0.186	**0.192**	**0.288**	**0.24**	**0.38**	**3.25**	**3.57**

Table 3.2: Image captioning results for selected random sampling and beam search methods. SPICE@1 measures the SPICE score of the most likely caption. SPICE@10 is the maximum score across the 10 candidates generated by each method. Mean SPICE is the mean score over all 10 candidates. In each section, the best value is bolded.

the candidate captions, which is useful to know for applications that combine diverse candidate output sequences [20].

Evaluating Diversity

To measure the diversity across the generated candidate sequences for a given input, we report **Dist-k**, the total number of distinct k-grams divided by the total number of produced tokens in all of the candidate responses for a prompt [26]. We report Dist-2 and Dist-4 averaged over the prompts in the test set.

A limitation of Dist-k is that all k-grams that appear at least once are weighted the same, ignoring the fact that infrequent k-grams contribute more to diversity than frequent ones. Zhang et al. [47] instead propose an entropy metric, **Ent-k**, defined as:

$$\textit{Ent-k} = \frac{-1}{\sum_{w \in S} F(w)} \sum_{w \in S} F(w) \log \frac{F(w)}{\sum_{w' \in S} F(w')}$$

where S is the set of all k-grams that appear in candidate responses for an example, and $F(w)$ denotes the frequency of w in the candidate responses.

3.3.4 Results

We report results on dialog systems and image captioning in Tables **??** and **??**, respectively. As expected, random sampling-based approaches yield outputs with greater diversity but worse quality than beam search-based approaches. Over-sampling then filtering increases the quality of outputs while still ensuring high diversity. In the following sections, we discuss the diversity-quality tradeoff, and then delve further into the results for each method group.

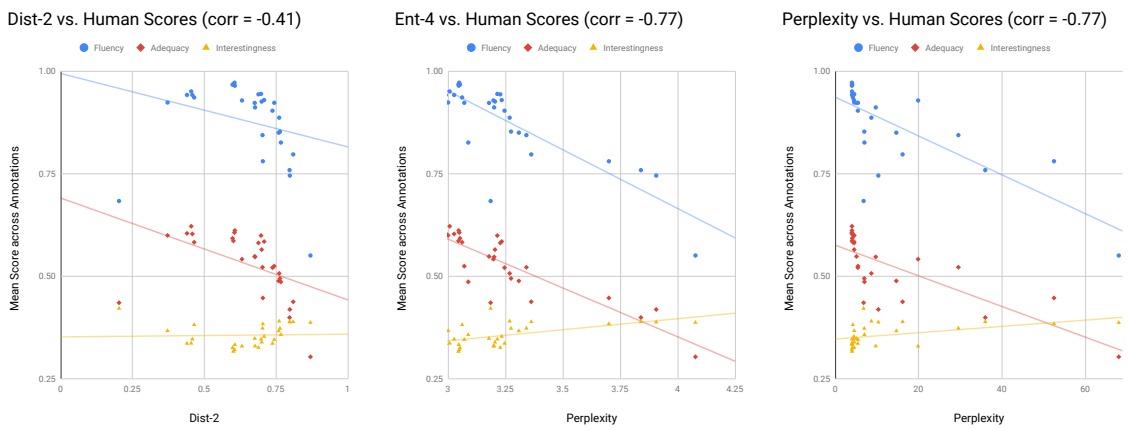


Figure 3.2: Each decoding strategy is plotted, showing that human-perceived quality is negatively correlated with diversity. The Pearson Correlation coefficients between each statistic and the average of fluency, coherence, and interestingness are shown in parentheses.

The Quality Diversity Tradeoff

The goal of diverse decoding strategies is to generate high-quality candidate sequences which span as much of the space of valid outputs as possible. However, we find there to be a marked trade-off between diversity and quality. This can be seen in Figure **??**, where we plot the human-judged quality score for each dialog experiment against our primary diversity descriptive statistics. Fluency and adequacy are both strongly negatively correlated with diversity. While we had expected interestingness to be positively correlated with diversity, the fact that it is not suggests that existing diversity statistics are insufficient for capturing what it means to humans for outcomes to be interesting.

Likewise, in image captioning, the mean SPICE score of the 10 candidate captions (averaged over all examples for each experimental setting) is strongly anti-correlated with diversity, with a Pearson correlation coefficient of -0.83 with the Ent-4 measure and -0.84 with Dist-2. Clearly it remains an open challenge to generate a diverse set of image captions that are all high-quality.

When researchers choose to use a diverse decoding strategy, they must decide where on the quality-diversity tradeoff they would like to lie; selecting an optimal method depends strongly on one's tolerance for errors. In machine translation, where mistakes could severely impact coherence, beam search-based methods, which tend to result in better fluency and coherence, but worse diversity might be preferred. In more open-ended applications, where novel text is of greater importance, increased diversity could be worth the fluency and coherency hit. As state-of-the-art models continue to improve, one would hope that the quality cost of encouraging diversity will continue to decrease.

In the interest of reporting a single overall best method for each task, we computed a sum-of-ranks score for each method. For dialog, we ranked the methods each by fluency, coherence, interestingness, and Ent-4, and then took a weighted sum of the four ranks, with 50% of the weight assigned to Ent-4, and 50% distributed evenly among the human evaluation ranks. Overall, clustered beam search and standard BS (beam size 100, PDC) have the best scores, followed by clustered beam search (beam size 10). Similarly, for image captioning, we rank the methods by their mean SPICE score and by Ent-4. Summing these ranks, random sampling (temp 1.0, top-10 capping, PDC) came in first. Standard beam search, Hamming Diversity beam search, and Top-g capping beam search (beam size 10) tied for second.

3.3.5 Random Sampling-based Methods

Higher sampling temperatures result in both an increase in diversity in generated responses and a reduction in overall quality. In the dialog domain, evaluators consistently rate the responses sampled with temperature 1.0 to have worse fluency, coherence, and interestingness when those sampled with temperature 0.5. In the image captioning do-

main, lower temperature improves automatic evaluation metrics for quality while reducing diversity.

For dialog, restricting sampling to the top-10 vocabulary words is a more effective strategy than adjusting temperature for ensuring balance between the quality and diversity of outputs. Top-10 random sampling has the highest fluency, coherence, and interestingness, as well as significantly lower perplexity than other random sampling methods. However, this trend did not extend to image captioning, where top-10 random sampling results in both worse SPICE scores and lower diversity measures than setting the temperature to 0.7. This may be because image captioning is a less ambiguous task than open-ended dialog, leading to a better-trained model that puts more probability mass on high-quality vocabulary words, ameliorating the challenge top-c filtering is designed to eliminate: that of a long tail of low probability vocabulary words taking up a large amount of probability mass.

Beam Search-based Methods

For dialog, clustered beam search (Cluster5 BS) performs the best of all beam search methods in terms of human-judged interestingness. It ties for best with NPAD0.3BS on fluency and ties with Standard BS on coherence. Iterative beam search (Iter5 BS) achieves the greatest diversity, but at the expensive of quality. It has the lowest human-judged coherence among beam search methods; thus, we do not evaluate this method on image captioning. For image captioning, Cluster5 BS has the highest diversity among beam search methods, but this difference is quite small. Cluster5 BS also has the highest SPICE@10 score, indicating it is the best method for generating at least one high quality candidate. However, Top3Cap BS results in the highest mean SPICE score, suggesting it is best at ensuring all outputs are reasonable quality.

Effect of Over-sampling

In our experiments, we explore over-sampling 100 outputs, and then either using post-decoding clustering (PDC) or re-ranking by log-likelihood to filter these 100 down to 10 diverse outputs.

In the dialog domain, this over-sampling approach is a definite win. When over-sampling with random sampling both methods of filtering substantially improve human judgements of fluency and adequacy compared to random sampling only 10 outputs. However, interestingness scores go down, and while the outputs are still more diverse than beam search-based methods, they are less diverse than random sampling without filtering. In the beam search methods that use a beam size of 100 then filter down to 10, human-judged quality is on par with beam size 10 results, but diversity is considerably higher.

When comparing the two types of filtering, PDC results in higher interestingness and diversity statistics, while log-likelihood re-ranking improves fluency and adequacy. This again demonstrates the trade-off between quality and diversity.[5]

For image captioning, over-sampling with reranking does not consistently improve quality as it does in the dialog domain. Mean SPICE score is improved for random sampling but not for beam search. SPICE@1 becomes worse for both random sampling and decoding, while SPICE@10 improves for random sampling, and for beam search when PDC is applied. From these results, we can conclude that over-sampling then ranking does not have a sizeable effect, either negative or positive, on quality. Moreover, the diversity of the captions generated by random sampling actually decreases when oversampling. The diversity of beam search-generated captions does improve with over-sampling.

5 In the appendix, we show results with every method where we generate 10 samples; generate 100 samples followed by selecting the 10 most likely outputs; and generate 100 samples followed by post-decoding clustering to select 10 outputs.

While oversampling does generally improve outcomes on the diversity/quality trade-off, it is more computationally expensive, particularly with beam search. Running PDC also requires generating sentence embeddings for every output, which adds additional computation time.

3.3.6 Summary of Contributions

The work described in this section was published in the 2019 Proceedings of the Association of Computational Linguistics [18]. The work was performed with in conjunction with Reno Kriz, with the assistance of Maria Kustikova and the mentorship of João Sedoc and Chris Callison-Burch. Reno and I contributed equally to the development of the ideas in the paper and the design and implementation of the experiments.

3.4 IMPACT OF DECODING STRATEGY ON THE DETECTION OF MACHINE-GENERATED TEXT

State-of-the-art generative language models are now capable of producing multi-paragraph excerpts that at a surface level are virtually indistinguishable from human-written content [**zellers2019defending**, **adelani2020generating**, 38]. Often, only subtle logical fallacies or idiosyncrasies of language give away the text as machine-generated, errors that require a close reading and/or domain knowledge for humans to detect.

Deceptive text, whether human- or machine-generated, has entered the sphere of public concern [**cooke2018fake**]. It propogates quickly [**vosoughi2018spread**], sets political agendas [**vargo2018agenda**], influences elections [**allcott2017social**], and undermines

user trust [**wang2012serf**, **song2015crowdtarget**]. Recently, **adelani2020generating** have shown that automatically generated reviews are perceived to be as fluent as human-written ones. As generative technology matures, authors, well-meaning or otherwise, will increasingly employ it to augment and accelerate their own writing. However, there has thus been little inquiry into the textual properties that cause humans to give generated text high human-like ratings compared to those that cause automatic systems to rate it highly.

This task of trying to guess whether text is coming from a robot or a fellow human was made famous by the Turing Test [**turing1950computing**]. It continues to be used is chatbot evaluation [**lowe2017towards**]. The related (but not identical) task of asking human raters to judge the quality of machine-generated excerpts remains the gold-standard for evaluating open-domain generation systems [**van2019best**].

In turns out that even with a fixed language model, choice of decoding strategy has a huge impact on the detectability of generated text. Using top-k random sampling, a decoding method where only the selection of high-likelihood words is permetted, means we are less likely to make a poor choice and create the type of mistakes that are easy for humans to detect. Since humans are not proficient at identifying when a model subtly favors some utterances more often than a human author would, they don't notice the over-representation of high-likelihood words in the generated text. In contrast, automatic systems excel at identifying statistical anomalies and struggle to build deeper semantic understanding. Top-k in particular creates text that is easy for machines to detect but very hard for humans. Thus, we observe the general trend: as the number of unlikely words available to be chosen is increased, humans get *better* at detecting fakes while automatic systems get *worse*.

In this work, we study three popular random decoding strategies—top-k, nucleus, and temperature sampling—applied to GPT-2 [38]. We draw a large number of excerpts

generated by each strategy and train a family of BERT-based [10] binary classifiers to label text excerpts as human-written or machine-generated. We find large differences in human rater and classifier accuracy depending on the decoding strategy employed and length of the generated sequences. Regardless of strategy, we find human raters achieve significantly lower accuracy than the automatic discriminators. We also show that when a decoding strategy severely modifies the unigram token distribution, as top-k does, humans have trouble detecting the resultant generated text, but automatic classifiers find it the easiest to discriminate. Worryingly, we further find that classifiers are brittle; they generalize poorly when trained to discriminate samples from one strategy and then evaluated on samples from another.

3.4.1 Detection as a Task

The rise of machine-generated content has led to the development of automated systems to identify it. GROVER was designed to not only generate convincing news excerpts but to also identify them using a fine-tuned version of the generative model itself [**zellers2019defending**]. GLTR, expecting attackers to use sampling methods that favor high-likelihood tokens, aims to make machine-generated text detectable by computing histograms over per-token log likelihoods [**gehrmann2019gltr**]. **bakhtin2019real** frame human-text detection as a ranking task and evaluate their models' cross-domain and cross-model generalization, finding significant loss in quality when training on one domain and evaluating on another. **schuster2019we** argue that the language distributional features implicitly or explicitly employed by these detectors are insufficient; instead, one should look to explicit fact-verification models. Finally, discriminators for whether text is machine-generated are a promising research direction in adversarial training

[**lin2017adversarial**, **li2017adversarial**] and in automatic evaluation of generative model quality [**novikova2017we**, **kannan2017adversarial**, **lowe2017towards**].

We frame the detection problem as a binary classification task: given an excerpt of text, label it as either human-written or machine-generated. In particular, we are interested in how variables such as excerpt length and decoding strategy impact performance on this classification task. We thus create several datasets. Each is approximately balanced between positive examples of machine-generated text and negative examples of human-written text. While they all share the same human-written examples, each dataset contains a different set of machine-generated examples sampled using one particular decoding strategy. We also build additional datasets by truncating all of the examples to a particular sequence length,

By training a separate classifier on each dataset, we are able to answer questions about which decoding strategy results in text that is the easiest to automatically disambiguate from human-written text. We are also able to answer questions about how the length of the examples in the training set impacts our ability to automatically classify excerpts of that same length as either human-written or machine-generated.

3.4.2 Method

Dataset Construction

All of our generated text samples are drawn from GPT-2, a state-of-the-art Transformer-based generative language model that was trained on text from popular web pages [38]. While we use the GPT-2 Large model with 774M parameters, we found that similar trends to those reported here hold in experiments with smaller language models.

Given an autoregressive language model that defines a probability distribution over the next token given the previous tokens in a sequence, a decoding strategy generates text by deciding how to output a token at each step based on the predicted distributions. Perhaps the most straightforward decoding strategy is to randomly choose a token with probability proportional to its likelihood. A challenge with the random sampling approach is that these probability distributions often contain a long tail of vocabulary items that are individually low-probability but cumulatively comprise a substantial amount of probability mass. **holtzman2019curious** observe that choosing tokens from this tail often leads to incoherent generations.

Top-k sampling, nucleus sampling, and (in the extreme) beam search have all been proposed to heuristically promote samples with higher per-token likelihoods. Top-k and nucleus sampling both do so by setting the likelihood of tokens in the tail of the distribution to zero. Top-k restricts the distribution to all but the k most likely tokens, where k is a constant [**fan2018hierarchical**]. Nucleus sampling, also called top-p, truncates the distribution at each decoding step t to the k_t-most-likely next tokens such that the cumulative likelihood of these tokens is no greater than a constant p [**holtzman2019curious**].

We thus consider three different decoding strategy settings:

- Sample from the untruncated distribution
- Top-k, choosing k=40 [38].
- Nucleus sampling (aka top-p), choosing p=0.96 [**zellers2019defending**].

In addition, we form “negative" examples of human-written text by taking excerpts of web text that come from the same distribution as GPT-2’s training data.[6] By picking text that resembles GPT-2’s train set, we ensure that our classifiers can’t simply take advantage of stylistic differences between the human-written text corpus and the kind of text GPT-2 was trained to generate.

6 https://github.com/openai/gpt-2-output-dataset

For each decoding method, we construct a training dataset by pairing 250,000 generated samples with 250,000 excerpts of web text. 5,000 additional paired samples are kept aside for validation and test datasets. Lastly, we filter out excerpts with fewer than 192 WordPiece tokens [**wu2016google**] (excerpts might be quite short if the model produces an end-of-text token early on). See Appendix 1 for final dataset sizes.

A crucial question when generating text with a language model is whether or not to provide a priming sequence which the language model should continue. Unconditioned samples, where no priming text is provided, in conjunction with top-k sampling, lead to pathological behavior for discriminators as the first token of the generated text will always be one of k possible options. On the other hand, if long sequences of human text are used as priming, the space of possible generated sequences is larger, but the detection problem shifts from one of "how human-like is the generated text?" to "how well does the generated text follow the priming sequence?".

Since in this study we are interested in the former simpler question, we create two datasets, one with no priming, and one with the minimum amount of priming possible: a single token of web text. This means that for every excerpt of web text in the training set, there is an excerpt of machine-generated text that starts with the same token. We find that even with limited priming, the ability of automatic detectors can be strongly impacted.

To study the effect of excerpt length, we construct variations of the above datasets by truncating all excerpts to ten possible lengths ranging from 2 to 192 WordPiece tokens [**wu2016google**]. In total, we obtain sixty dataset variations: one per sampling method, truncation length, and choice of priming or no priming.

Methods for Automatic Detection

The primary discriminator we employ is a fine-tuned BERT classifier [10]. We fine-tune one instance of BERT per dataset variation described above. For the longest sequence length, n=192, we compare BERT's performance with several simple baselines that have been proposed in other work.

FINE-TUNED BERT We fine-tune BERT-LARGE (cased) on the task of labeling a sentence as human- or machine- generated. The models are trained for 15 epochs, with checkpoints saved every 1000 steps, and a batch size of 256. All results are reported on the test set using the checkpoint for which validation accuracy was highest.

BAG-OF-WORDS For each sequence, we compute a bag-of-words embedding where each dimension corresponds to a token in GPT-2's 50,000 token BPE vocabulary [40], and we count how many times that token appears in the text sequence. We then train a logistic regression binary classifier to predict human- or machine-written given this 50,000-dimensional embedding. We experimented with truncating embedding size by removing entries for infrequent vocabulary words, but this did not improve performance.

HISTOGRAM-OF-LIKELIHOOD RANKS Following GLTR [**gehrmann2019gltr**], we compute the probability distribution of the next word given the previous words in a text sequence according to a trained language model (in our case the same GPT-2 model that was used for generation). At each sequence position, we rerank the vocabulary words by likelihood, and record the rank of the ground-truth next word within this list. These ranks are then binned. GLTR uses four bins, counting (1) the number of times the top 1 word is seen, (2) the number of times words ranked 2 through 5 are seen, (3) words ranked 6-100, and

(4) words ranked >100. However, we observe higher accuracy when 50 bins are spread uniformly over the possible rankings. This means that since there are 50,000 vocabulary words, the first bin counts the number of times the actual next word was within the 1,000 mostly likely next words, the second bin counts the 1,001-2,000th, and so on. We then train logistic regression binary classifiers to predict human- or machine-written given either the 4-dimensional histograms or 50-dimensional histograms as input.

TOTAL PROBABILITY **solaiman2019release** propose a very simple baseline consisting of a threshold on the total probability of the text sequence. An excerpt is predicted as machine-generated if its likelihood according to GPT-2 is closer to the mean likelihood over all machine-generated sequences than to the mean of human-written ones.

Method for Human Detection

The human evaluation task is framed similarly to the automatic one. We ask the raters to decide whether a passage of text was written by a human or by a computer algorithm. (Full instructions are in the Appendix.) Raters are allowed to choose between four options: “definitely" or “possibly" machine-generated and “definitely" or “possibly" human-written. They are first shown an excerpt of length 16 WordPiece tokens. After they make a guess, the length of the excerpt is doubled, and they are asked the same question again. This continues until the entire passage of length 192 tokens is shown. Passages are equally likely to be human-written or machine-generated, with the machine-generated excerpts being evenly split between the three sampling strategies considered in this paper.

Initially, Amazon Mechanical Turk (AMT) raters were employed for this task, but rater accuracy was poor with over 70% of the “definitely" votes cast for “human" despite the classes being balanced. Accuracy, even for the longest sequences, hovered around 50%. The same study was then performed with university students who were first walked

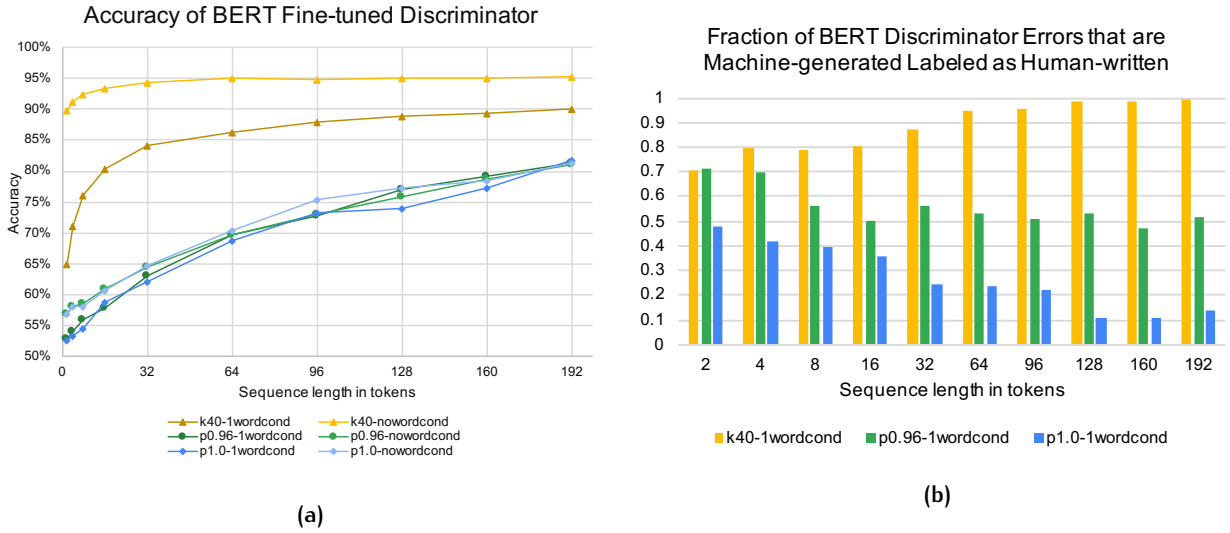


Figure 3.3: In **(a)**, accuracy increases as the length of the sequences used to train the discriminator is increased. In **(b)**, we see that the BERT fine-tuned discriminator predicts about the same number of false-positives as false-negatives when trained with samples generated using top-p sampling. However, for top-k, it more often mistakes machine-generated text to be human-written, while for untruncated random sampling the opposite is the case.

through ten examples (see Appendix Table 4) as a group. Afterward, they were asked to complete the same tasks that had been sent to the AMT workers. No additional guidance or direction was given to them after the initial walk-through. We will refer to this group as the "expert raters." Among them, 52.1% of "definitely" votes were cast for human, and accuracy on the longest excerpt length was over 70%.

The human evaluation dataset consisted of 150 excerpts of web text and 50 excerpts each from the three decoding strategies. Each question was shown to at most three raters, leading to 900 total annotations from the untrained workers and 475 from the expert raters. A more detailed breakdown can be found in the Appendix.

3.4.3 Results

SIMPLE BASELINES Table **??** shows the performance of the baseline discriminators on length-192 sequences, as compared with fine-tuned BERT. Reassuringly, BERT far surpasses all simple baselines, indicating that it is not fully possible to solve the detection problem without complex sequence-based understanding. The simplest baseline, TotalProb, which makes a decision based on the likelihood of the sequence, performs surprisingly well (over 60% accuracy for all sampling methods) relative to the methods which involve training logistic regression models.

Logistic regression on bag-of-words is the best of the baselines, beating out the histogram-based methods. While **gehrmann2019gltr** report an AUC of 0.87 on classifying text as real or generated using logistic regression on the four buckets of the GLTR system, we report AUC between 0.52 and 0.56 for this task. The discrepancy is likely due to the fact that the human-written text in our discriminator training set comes from the same distribution as the text used to train the language model, while in GLTR the human text comes from children's books, scientific abstracts, and newspaper articles. The selection of training data for learned detection systems is crucial. In real-world applications, the choice ought to reflect the genres that builders of text-generation systems are trying to impersonate.

FINE-TUNED BERT In Figure **??**, we begin by observing discriminator accuracy as a function of excerpt length and sampling method. As can be intuitively expected, as sequence length increases, so too does accuracy. For unconditioned text decoded with nucleus (p0.96) and untruncated (p1.0) random sampling, we find discriminator accuracy increases from 55%, near random, to about 81% for the longest sequences tested. In

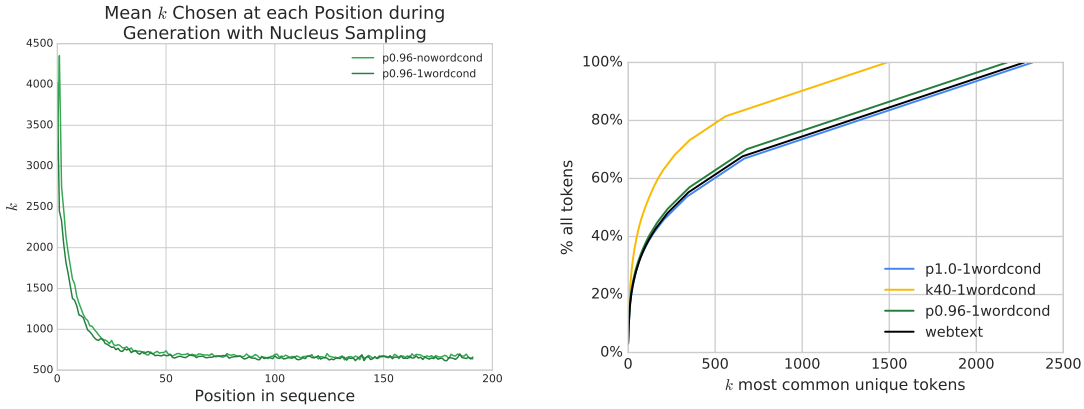


Figure 3.4: In **(a)**, the average (over sequences in the test set) k chosen at each step during generating with nucleus sampling is plotted. Adding a single word of priming strongly impacts the ks chosen for the first few positions, but this difference quickly dissipates. In **(b)**, we consider the first token generated in each sequence by top-k, and plot what fraction of these are captured by the k most common unique tokens from the vocabulary. Overall, at its first step, top-k concentrates 80% of its probability mass in the 500 most common tokens from the vocabulary.

contrast, discriminators trained and evaluated on top-k achieve over 80% accuracy even on 16-token excerpts.

Why are top-k's samples so easy to detect? In Figure **??**, we see the percentage of probability mass concentrated in the k most common token types for each sampling method. While random sampling and nucleus sampling are very similar to human-written texts, we see top-k concentrating up to 80% of its mass in the first 500 most common tokens. The other sampling methods as well as human-written texts require at least 1,100 token types for the same. It is clear that top-k's distribution over unigrams strongly diverges from human-written texts–an easy feature for discriminators to exploit. In fact, **see2019massively** note that it takes setting k to 1000 to achieve about the same amount of rare word usage and fraction of non-stopword text as as human writing.[7] This makes it

7 when decoding from the GPT-2 small model with 117M parameters.

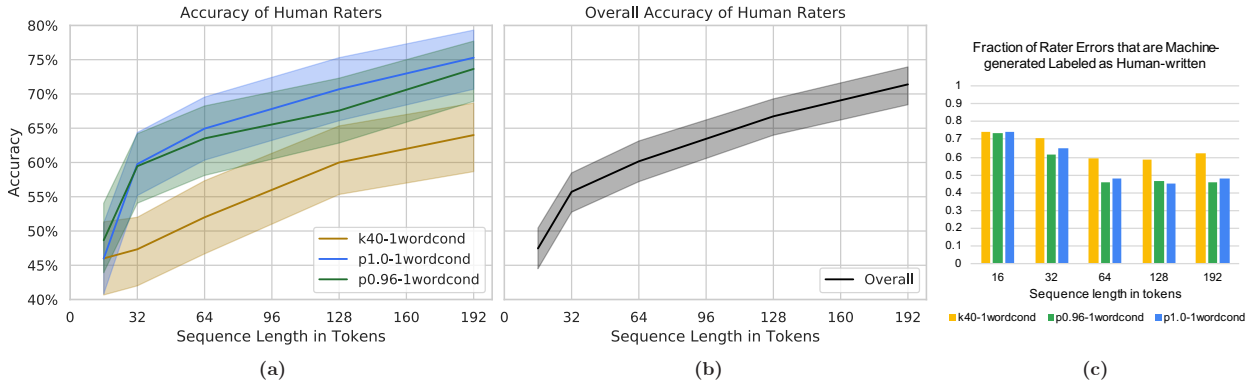


Figure 3.5: **(a)** and **(b)** show human rater accuracy of correctly identifying an excerpt as human-written or machine-written, shown with 80% confidence internals, in **(a)**, broken up by decoding strategy and in **(b)**, overall. Accuracy increases as raters observe more tokens. **(c)** shows that for short excerpts, most rater mistakes are them incorrectly thinking machine-generated text is human written. The two errors types become more balanced at longer lengths.

very easy for the model to pick out machine-generated text based on these distributional differences.

One way to help resolve this problem is to add priming text. Doing so causes more rare words to be incorporated into the top-k of the unigram distribution. Adding even a single human word of priming significantly reduces the performance of detectors trained with top-k random sampling. Without priming, a discriminator trained on sequences of length 2 can classify with ∼90% accuracy the provenance of the text (Figure **??**). By adding one priming token, accuracy drops to ∼65%. Even on the longest 192-length sequences, top-k discriminator accuracy is 6% lower on the primed dataset than the unprimed one.

When generating with nucleus or untruncated random sampling, adding a priming token is not as impactful, as these methods are already sampling from a large fraction (or all) of the probability distribution. This is seen in Figure **??** where at the very first step of unprimed generation, nucleus sampling selects from 3075 possible vocabulary words, and at later positions selects from on average more than 500. Untruncated random sampling always selects from the entire 50,000 word vocabulary, whereas top-k only selects from k.

TRANSFERABILITY In Table **??**, we show how discriminators trained with samples from one decoding strategy can transfer at test time to detecting samples generated using a different decoding strategy. Unsurprisingly a discriminator trained on top-k generalizes poorly to other sampling methods: accuracy drops to as low as 42.5%, *worse than chance*. Conversely, training the discriminator with sequences sampled from the untruncated distribution leads to little transferability to detecting top-k samples. Only the discriminator trained with nucleus sampling (a compromise between unmodified sampling and top-k) was able to detect sequences from the other sampling strategies without too much of a hit to accuracy. As expected, a discriminator trained on an equal portion of data from each decoding method does reasonably at detecting all three.

Perhaps this lack of transferability is related to each discriminator's calibration. Indeed, the degree to which a discriminator's average prediction deviates from 50% is a direct indicator of its accuracy. In Table **??**, we observe that of the three BERT discriminators, only that trained on top-p samples predicts 'machine-generated' on approximately 50% of in-domain examples as expected. This same discriminator's behavior holds on datasets generated by other sampling strategies as well. In contrast, we observe that discriminators trained on top-k and untruncated random samples severely underestimate the percentage of machine-generated excerpts in out-of-domain datasets. Even within domain (Figure **??**), we find both discriminators heavily favor a single class, increasingly so as the number of tokens increases.

HUMAN EVALUATION Overall human performance across all sampling methods is shown in Figure **??**b. Even with the multi-paragraph 192-length excerpts, human performance is only at 71.4%, indicating that even trained humans struggle to correctly identify machine-generated text over a quarter a time. However, it is worth noting that our best raters achieved accuracy of 85% or higher, suggesting that it is possible for humans to do very

well at this task. Further investigation is needed into how educational background, comfort with English, participation in more extensive training, and other factors can impact rater performance.

To break up the accuracies by sampling method in a way that is comparable to the results shown for the automatic discriminators, we pair each machine-generated example with a randomly selected one of webtext to create a balanced dataset for each sampling strategy. Performance is shown in Figure **??**a. Top-k produces the text that is hardest for raters to correctly distinguish, but as shown in Section **??**, it is the easiest for our automatic detection systems. Samples from untruncated random sampling and nucleus sampling with p=0.96 are equivalently difficult for raters to classify as machine-generated. Our human evaluation results suggest that much lower p-values than the 0.92 to 0.98 range proposed in **zellers2019defending** might be necessary in order to generate text that is considered significantly more human-like to human raters than the text produced by using the untruncated distribution.

Table **??** gives several examples where human raters and our BERT-based discriminators disagreed. When raters incorrectly labeled human-written text as machine-generated, often the excerpts contained formatting failures introduced when the HTML was stripped out. In the middle two examples, topic drift and falsehoods such as Atlanta being the “information hub of the nation’s capital" allowed humans to correctly detect the generated content. However, in the bottom two examples, the high level of fluency left human raters fooled.

Overall we find that human raters—even “expert" trained ones—have consistently worse accuracy than automatic discriminators for all decoding methods and excerpt lengths. In our experiments, randomly-selected pairs of raters agree with each other on a mere 59% of excerpts on average. (In comparison, raters and discriminators agree on 61% to 70% of excerpts depending on the discriminator considered). We surmise that the gap between

human and machine performance will only grow as researchers inevitably train bigger, better detection models on larger amounts of training data. While improved detection models are inevitible, it is unclear how to go about improving human performance. GLTR proposes providing visual aids to humans to improve their performance at detecting generated-text, but it is unlikely that their histogram-based color-coding will continue to be effective as generative methods get better at producing high-quality text that lacks statistical anomalies.

3.4.4 Summary of Contributions

The work described in this section was published in the 2020 Proceedings of the Association of Computational Linguistics [17]. The work was performed with in conjunction with Daniel Duckworth, with the mentorship of Douglas Eck and Chris Callison-Burch. I proposed the idea of studying detection, and Daniel and I worked both worked on designing and implementing the experiments and analyzing the results.

3.5 ROFT: A LARGESCALE STUDY OF HUMAN DETECTION ABILITY

3.5.1 Introduction

In the previous section, we conducted a small-scale study showing how choice of decoding strategy impacts human ability to detect machine-generated text. However, there are many other factors which influence detectability that we were not able to include in this study, including the domain of the text being used for evaluation and the architecture and manner in which the underlying language model was trained. In addition, we were

interested in studying the annoators themselves–how do annotator background as well as the incentive structure set up for soliciting annotations impact performance on the detection task? We therefore saw the necessity of designing a platform for conducting large-scale studies of the detection task.

Existing studies, including the one in Section **??**, focused on the binary question of whether or not a provided document contains any generated text. For our large-scale study, we instead framed detection as a boundary-detection task: given a document that starts off as human-written and at some point transitions to machine-generated, can annotators detect the transition point? The boundary detection setting is more informative than the classification setting because it better aligns with how LMs are used to generate text in practice; in typical usage, a generative system is provided with a prompt and asked to produce a continuation. By measuring human skill at the boundary detection task, we were able to evaluate the relative performance of different generative systems, build a better understanding of how incentive structure influences the quality of the annotations acquired, and make progress toward quantifying the risks associated with large language model goals.. Furthermore, because the annotation platform we built was public, we could achieve these research goals while simultaneously educating the public about how to spot generated text.

The primary contributions of this section are as follows:

- The “Real or Fake Text” game, which anyone can play on the internet.
- Large-scale case study on how a game-like platform can be used to perform data collection for the detectability task.
- Analysis of how generation-time design choices impact detectability of generated text.
- Study of the errors and text properties which humans associate with machine generations.

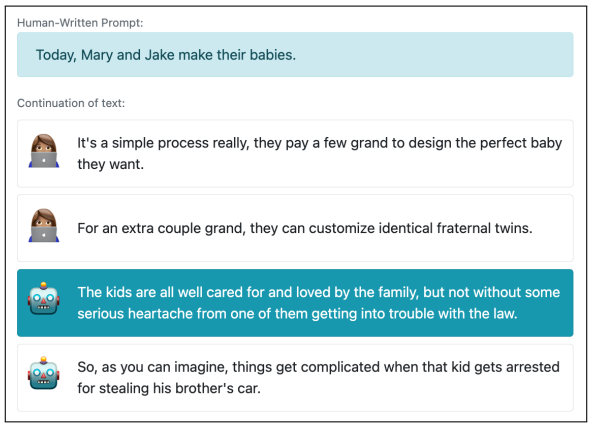


Figure 3.6: In the boundary detection task, players see one sentence as a time and try to guess when they transition from human-written to machine-generated.

- Public dataset of 21,646 human annotations on the boundary detection task.

3.5.2 Related Work

Previous research on understanding the ability of humans to detect machine generated text has mostly posed the task as a classification task–given a text example that is either entirely human-written or entirely machine-generated (aside from an initial prompt), annotators must predict whether it is human-written or machine-generated. On this task, **ippolito-etal-2020-automatic** reported that trained evaluators were able to achieve an accuracy of at best 71.4%, using generation from GPT-2 Large [38]. **clark2021all** demonstrated that annotators are able to distinguish GPT-2 XL generations with at best 62% accuracy, but they perform no better than random chance on GPT-3 outputs [**brownetal2020**]. Even after training evaluators to improve their detection abilities, detection accuracy on GPT-3 was only able to converge to around 55%. A study by **brownetal2020** reported similarly low performance (52%) on the detection of machine-generated news articles.

Our work builds directly off of the RoFT detection game introduced by **roft**; we contribute expanded analysis across a variety of genres and generative systems. A similar detection task, where annotators guess whether turns in a conversation were generated, was posed as a way to evaluate dialog systems [**deriu-etal-2020-spot**].

Another related area of research is asking annotators to explain why they think generated text is generated. **he-etal-2021-tgea** created a dataset of generated text annotated with the errors that humans found in it, and **dou2021scarecrow** proposed an error annotation schema for generated text. The errors we allow players to report in our experiments were inspired by this schema.

3.5.3 The Real or Fake Text Game

In order to facilitate data collection, we follow **roft** and pose the detection task as a game. In each game round, players were shown one sentence at a time and earned points for guessing close to the true boundary (Figure **??**). They were also asked to select reasons for why they made their decision. In total we collected over 20,000 annotations. We find that players vary substantially in their detection ability, and that factors such as the amount of time taken to complete a game round and total number of game rounds played sometimes correlate with success. We discuss the difficulty in incentivizing players to improve over time. Furthermore, we examine some of the the trends and errors which distinguish real from generated text and look at whether annotators could pick up on these trends.

Our study uses data collected through the "Real or Fake Text" (ROFT) annotation platform [**roft**]. ROFT is a turn-based game where a player first selects a domain of text (news articles, recipes, short stories, or speeches). The player then plays a series of game rounds. In each round, the player is shown a starting sentence which they are told comes from a real human-written document. They are then shown subsequent sentences, one at

a time. Each subsequent sentence may be the true continuation of the document, or it may be text generated by a language model. Once the sentences transition to being machine-generated, they will stay so for the rest of the 10-sentence passage.

After being shown each sentence, the player must guess whether that sentence was machine-generated or human-written. If the user selects "human-written," another sentence is displayed. If the player deems the current sentence to be written by a machine, the game round ends and the true author (machine or human) for each sentence is revealed. Before submitting their selection, the player is able to select a reason to explain their choice of sentence[8]. Thus, the player's goal in ROFT is to correctly identify the sentence at which a passage transitions from being human written to being generated by a language model. This setting is considerably more realistic than prior work, since in the real world, generating with a prompt is the standard way to achieve controllability, and malicious actors will not reveal what portion of a generation is the human-written prompt.

3.5.4 Experimental Design

Datasets

In order to answer questions of how textual genre and writing style affect detectability of machine-generated text, we selected four diverse categories of prompts. For each category, documents were sentence-segmented, and only documents with 11 or more sentences were retained. For each document, the first h sentences were used as the prompt, where h is a uniform random number between 1 and 10 (inclusive). The remaining $10 - h$ sentences of each 10-sentence game round were a machine-generated continuation. Our four genres of prompts are as follows:

8 See Table **??** in the Appendix for a full list of reasons given with their accompanying description text.

NEWS ARTICLES. Documents were drawn from the New York Times Annotated Corpus [**sandhaus2008new**], which contains 1.8 million articles published by the Times between 1987 and 2007. Our hypothesis was that this domain would be challenging for models since news requires factual accuracy, which state of the art models have been shown to struggle with [**nakano2021webgpt**, **lin2021truthfulqa**].

PRESIDENTIAL SPEECHES. Documents were drawn from the presidential speech corpus [**brown2016cops**], which contains 963 speeches given by presidents of the United States, with dates ranging from 1789 to 2015. Our hypothesis was that the sort of first-person rhetoric found in these speeches would be easy for models to impersonate since political speech and first-person speech are plentiful in web-based training data.

STORIES. Fictional stories were selected from the Reddit Writing Prompts dataset [**fan2018hierarchical**], a corpus of amateur short stories scraped from the r/Writing-Prompts sub-Reddit[9]. We hypothesized that this domain would be challenging for players since the writing quality of the stories is not especially high (which lowers the bar for the model generation quality), and factuality is not as important in a fictional domain.

RECIPES. Recipes were extracted from the Recipe1M+ dataset [**marin2019learning**]. Recipes were parsed slightly differently than the other domains. We set the "first sentence" of each document as the name of the recipe and the ingredient list, and each subsequent "sentence" was a step in the recipe. Some recipe steps were more than one sentence. We hypothesized that this dataset would be difficult for models due to the closed-ended nature of the task and the reliance on common sense.

9 https://www.reddit.com/r/WritingPrompts/

Awarding Points

In each game round, the player is awarded points based on how close their selection was to the true boundary[10]. Players were awarded 5 points for correctly choosing the boundary sentence and $\max(5-n,0)$ points for a guess n sentences after the boundary. Players were not awarded points for guessing a sentence before the boundary. Players were able to see how many points they earned in each category on their profile page and compare their performance with fellow players on the leaderboard page. In the Findings section (Section **??**), we report mean score earned as the predominant evaluation metric. This metric has high correlation with other more standard metrics (Appendix **??**).

Player Recruitment and Annotation Filtering

Players were recruited from two sections of an Artificial Intelligence course for Master's students and senior undergraduates at an American university. We only analyze fully anonymized data from students who consented to having their annotations used for research purposes.

The first section (Group A) was asked to play 30 minutes of the ROFT annotation game for a fixed amount of points of class credit. Students in this section were not given any instructions beyond how to create an account. The second section (Group B) was explicitly told they would be awarded $\min(p/250,2)$ points of extra credit toward their final grade, where p was the number of points the student earned on the ROFT leaderboard. Students in Group B were given detailed instructions[11] and examples of signs to look out for that text was machine-generated. Table **??** gives statistics on the annotations collected from each class.

10 For our purposes, the "boundary" sentence is considered to be the first machine-generated sentence in the passage

11 https://storage.cloud.google.com/roft_datasets/boundary_detection_guide.pdf

We note that university students taking an advanced artificial intelligence course are not reflective of the global population of English speakers, and the results presented in this paper may not reflect the general population's ability to detect machine-generated text.

In total, we collected 42,165 annotations over 7,895 different game rounds. The annotations were then filtered in the following ways. If a player guessed the same boundary position for a series of 5 or more rounds in a row, we removed all the annotations in the series because the player was likely no longer actually playing the game as designed. We also removed annotations from the two players cheated by exploiting Javascript vulnerabilities. Finally, for the recipes genre, a bug during dataset curation resulted in an over-representation of "all-human" game rounds played; for better balance during analysis, we randomly removed a portion of these annotations. More details on filtering are in Appendix **??**. Our final filtered dataset consisted of 21,646 annotations over 7,257 game rounds. For News, Stories, and Recipes, we had on average over 2 annotations per game round, while for Speeches, a smaller dataset, we had on average 16. Table **??** gives a detailed breakdown of the dataset across genres and generation systems.

Continuation Sources

In order to answer questions related to how model attributes affect generated text we employed different methods of text generation for each category. For Recipes, New York Times, and Stories, we generated continuations with GPT-2 XL using nucleus sampling [**holtzmanetal2020**] with $p = 0.4$ and a repetition penalty of 1.2 [**keskar2019ctrl**]. For Recipes, we additionally generated continuations with a GPT-2 XL model finetuned on recipes[12].

12 See Appendix **??** for finetuning details.

For New York Times and Stories, we experimented with varying the p used for decoding, testing out $p = 0.0$ (argmax) and $p = 1.0$ (sampling directly from the model's predicted distribution).

We also added 100 attention-check game rounds (10 for each possible context length), where rather than transitioning to machine-generated text, the passage transitioned to sentences from a randomly selected news article. We refer to this as the "random" model in our analysis and expect these game rounds to be trivial for players.

For Stories, we experimented with different model sizes, generating continuations with both GPT-2 Small (117M Parameters) and GPT-2 XL (1.5B Parameters). Lastly, for Presidential Speeches, we generated continuations using the CTRL model [**keskar2019ctrl**] rather than GPT-2. CTRL has the option to specify a control code indicating what domain to generate text in. For half of the generations, we used the "[Politics]" control code while for the other half we randomly selected a control code each time. Table **??** gives the statistics of the generations included. Note that not all generations received annotations.

3.5.5 Results

The collected annotations allow us to investigate several questions. Error bars on all figures and tables are 95% confidence intervals. Appendix **??** gives the exact values for all bar graphs.

Can humans detect generated text?

Players correctly guessed exactly on the boundary sentence 23.4% of the time. For game rounds which contained at least one generated sentence, players were able to eventually guess machine-generated 72.3% of the time, even if they missed the exact boundary.

Players incorrectly identified 61.3% of all-human game rounds as containing machine-generated text .

The average number of points (§**??**) received per round by our players was 2.08, well above random chance. As baselines, if a player uniform randomly guessed every round, their expected per-round score would be 1.31, and if they always guessed the last sentence, their expected per-round score would be 1.5.[13] For the remaining analyses, we will use average points earned as the primary measure of detection ability. This measure correlates with other possible metrics (Table **??**).

As an additional sanity check on annotator skill, we included game rounds in the News domain where instead of transitioning to an LM-generated continuation, the text transitioned to a completely different news article selected at random. Out of the 214 annotations we collected for this "sanity check" system, the mean score was 2.75, significantly higher than any of the true LM-backed systems. Also, for these annotations, the error type "irrelevant" was selected about twice as often as all other error types combined, validating that players were paying attention to the task at hand.

How much does player ability vary?

There was a large variance in the skill of individual players. Out of the 116 players who completed 50 game rounds, 19 earned a total score of 70 or fewer points (one std below the mean score) in their first 50 rounds, while 15 earned a total score 127 or greater points (one standard deviation above the mean score). Four of these raters scored two standard deviations above the mean score.

We also found that under the right conditions, players can improve over time. There was no correlation between number of rounds played and player score for Group A. However,

13 These expectations assume that the true boundary position is equally likely to be at any position. Figure **??** shows the true distribution of boundaries, which was not quite uniform.

Group B, who were given extra credit proportional to their game score, did show slight improvement (Table **??**). Interestingly, there was also lower variance in points earned among Group B (Figure **??**).

We can also measure inter-annotator agreement with the Krippendorff's alpha coefficient. This statistic measures how much disagreement there is between players compared to the amount of disagreement one would expect by chance. Two players are considered to have agreed if they both guessed "machine-generated" on any sentence after the true boundary or if they both guessed the entire passage was human-written. Over all annotations, we found α=-0.25, indicating there was less agreement than could be expected from random guessing, suggesting different annotators were better at identifying different kinds of problems with LM-generated text. However, among our top 25% of players (measured by mean score), there was high inter-annotator agreement, with α=0.44, suggesting that good annotators made similar errors.

Are some genres easier to detect?

We found that generated text was easier to identify in the recipes and speeches genres than in the stories and news genres. Figure **??** (left) shows the average points received on each genre for game rounds that used comparable LMs, while Table **??** gives a more detailed breakdown across models.

For recipes, we expect that the task was made easier by the fact that the first human-written "sentence" in each game round was a semi-structured ingredients list, making it easy for players to check for contradictions—a step saying to mix in cream is probably generated if there is no cream ingredient. In addition, recipes often assume implicit unwritten knowledge, which language models struggle to get right—a step saying to crack eggs into a bowl must precede a step saying to whisk the eggs. Indeed, if we look at the reasons given by our players for saying "machine-generated," recipes had a much larger

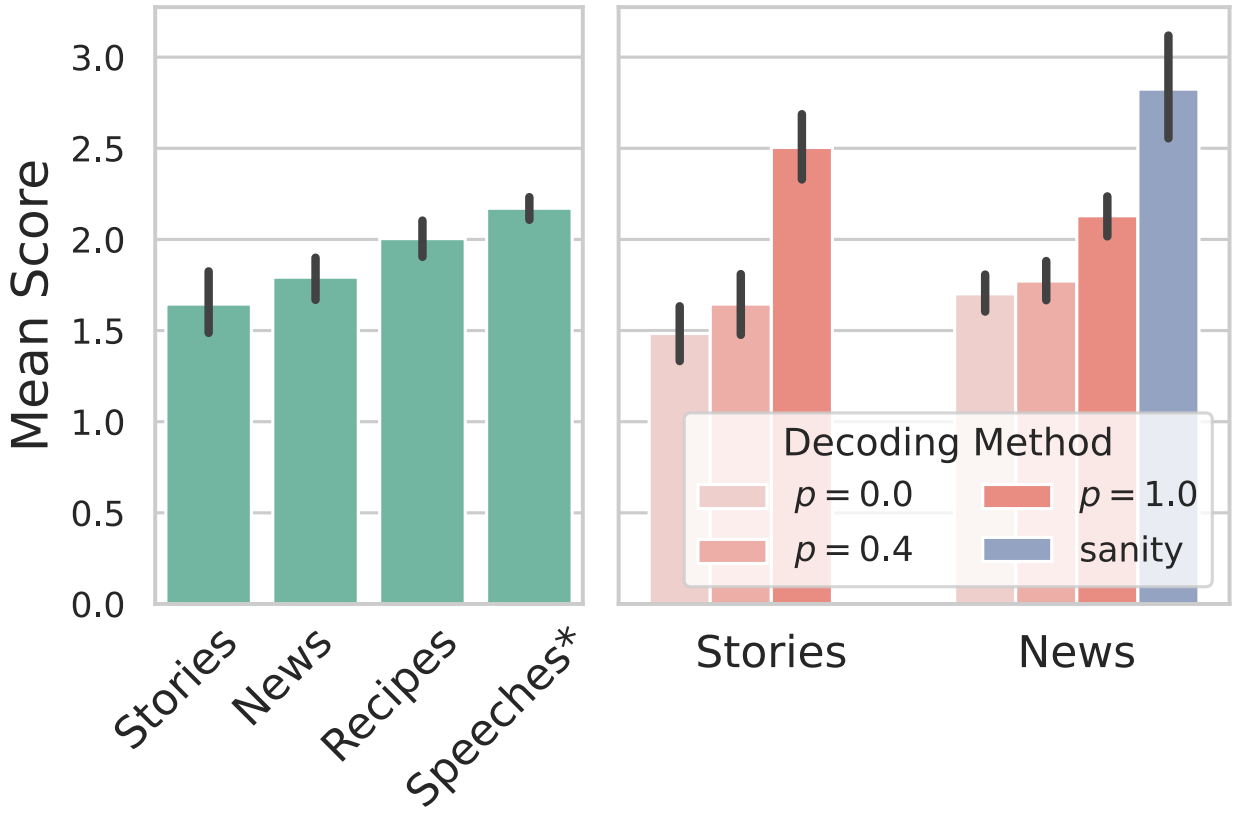


Figure 3.7: **(left)** Comparison of mean player score across different genres with GPT-2 XL p=0.4 (and CTRL p=0.4 for speeches). **(right)** Comparison of mean player score across different values of p for nucleus sampling (GPT-2 XL), as well as a "sanity-check" baseline.

percentage of "common_sense" errors (26%) than did either News (10%) or Stories (10%). It is worth noting that this result slightly contradicts the one reported by **clark2021all** who reported that generated recipes were more difficult to detect than news or stories; more targeted research is necessary to fully understand the relationship between domain and generation performance.

We believe the speech genre was easier for players not because speeches are intrinsically more difficult to generate but because we struggled to get CTRL to produce high-quality, non-repetitive generations, even though it is about the same size model as GPT-2 XL. It

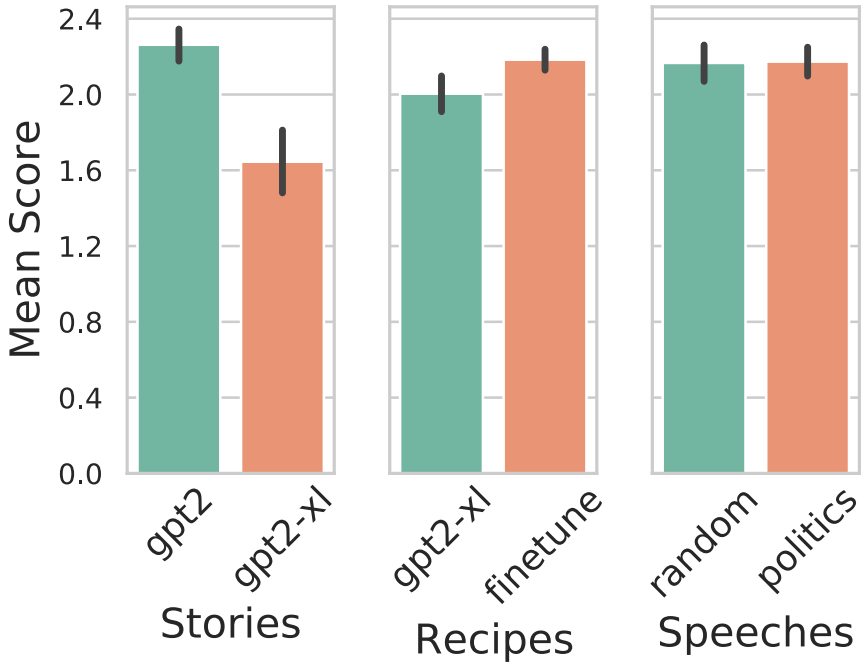


Figure 3.8: **(left)** For Stories, as model size increases (using p=0.4), detection becomes harder. **(middle)** For Recipes, extra finetuning does not significantly impact detectability. **(right)** For Speeches, using a "[Politics]" control has no impact on detectability.

was necessary to incorporate repetition penalties during generation with CTRL, which helped but did not solve the quality issues.

3.5.6 Does model size make a difference?

Previous work has shown that language model performance scales with number of parameters [**kaplan2020scaling**], so we expected players to be worse at detecting generations from larger models. Indeed, we found that players scored significantly higher when generations came from GPT-2 small (117M parameters) than when they came from GPT-2 XL (1.5B parameters) (Figure **??**.).

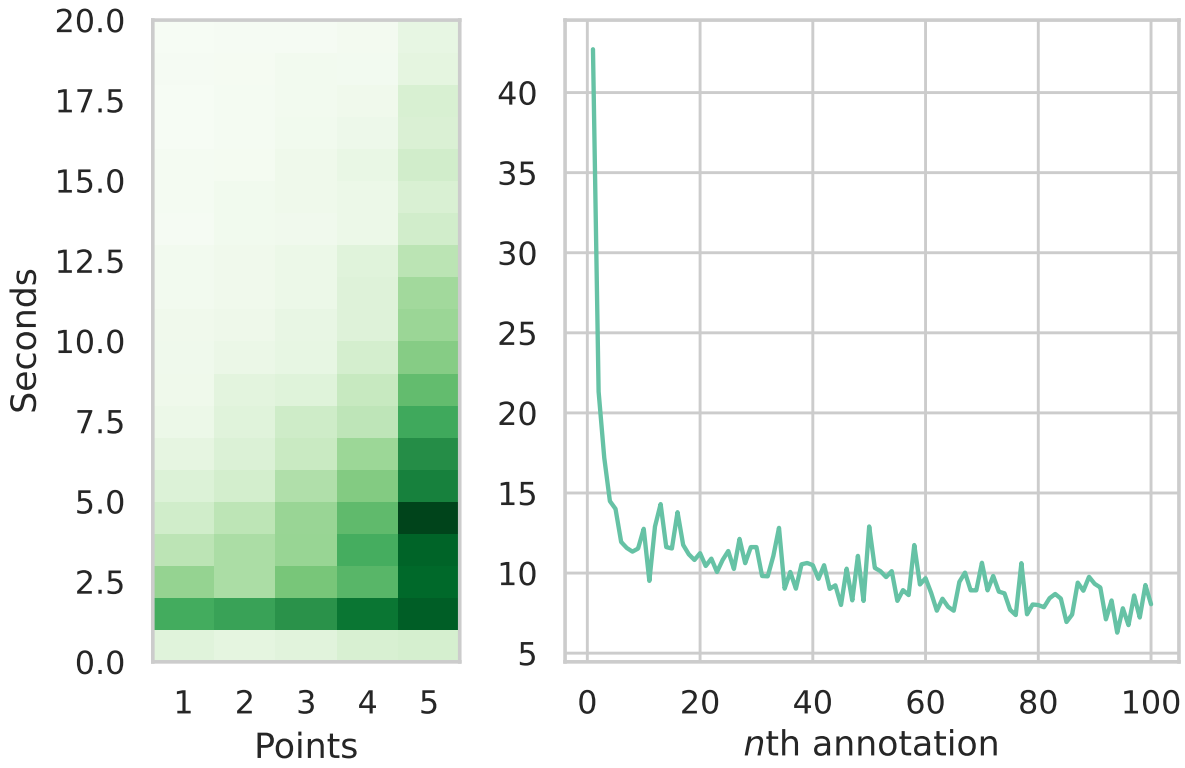


Figure 3.9: **(left)** Histogram showing the relationship between points earned and the number of seconds an annotation took. **(right)** Among players who completed at least 100 annotations, average annotation speed decreased with increased experience at the task.

3.5.7 Are diverse generations easier to detect?

Choice of decoding strategy is known to have significant impact on text quality [**zhang2020trading**] and detectability [**ippolito-etal-2020-automatic**]. Choosing a lower value of p when generating with a nucleus sampling [**holtzmanetal2020**] decoding strategy produces less diverse but also less noisy text than choosing a higher value of p. In our experiments, we did not find statistically significant differences in player skill between p=0.0 (greedy) and p=0.4 sampling (Figure **??**). However, players were significantly better at p=1 (pure random sampling) than the lower values, validating claims from earlier papers that LMs struggle

to generate high-quality text with similar diversity to human-written text. Interestingly, generations from GPT-XL using p=1.0 were easier for players to detect than generations from GPT-2 small using p=0.4. This highlights the importance of decoding, as improper selection of decoding strategy may cause a language model to perform worse than one that is one tenth its size.

3.5.8 Do control codes affect detectability?

CTRL is a 1.6B parameter LM trained with controllability in mind. At inference time, one can pass in a control code, such as "[Politics]" or "[Horror]" to include the style of the generated text. We investigated the efficacy of these control codes on the genre of presidential speeches by using "[Politics]" for half the generations and randomly selecting control codes for the remaining half. We found that use of the politics control code did not significantly affect players' ability to distinguish real from fake text. This is not to say that control codes do not affect generation; however, it does suggest that the cues used by players to detect generations may not be related to genre-specific details, as least not within the genre of political speeches. Further work is needed to investigate whether control codes could have influenced detectability in other genres.

3.5.9 Does finetuning affect detectability?

We had expected that finetuning on in-domain text would result in a model that was better able to fool humans. Counter to expectations, there was a small increase in player detection ability when generations came from GPT-2 finetuned on recipes compared with generations from pre-trained GPT-2. This is despite the fact that the finetuned model had

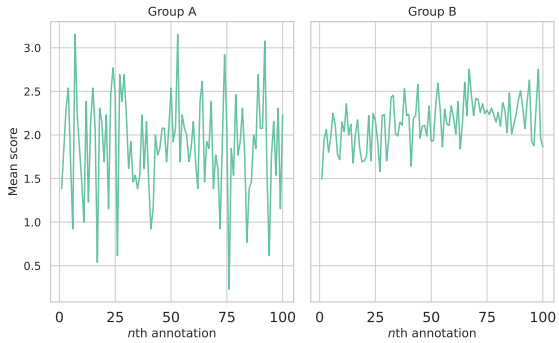


Figure 3.10: Performance over time for the two player groups (§**??**). Players in Group B, who were given extra instruction and incentives, improved over time while those in Group A did not.

close to half the perplexity of the pre-trained model on a held out test set of 50,000 recipes (4.781 vs. 8.979). While we can only speculate as to the amount of recipe knowledge present in the pre-trained model (GPT-2's training data is not publicly available), it is possible the pre-trained model already contained enough understanding of recipe-like text that it was not critical to do the extra-finetuning. Perhaps finetuning would have had more impact in a specialized or jargon-laden domain (e.g. legal, medical).

3.5.10 How much time did game rounds take?

To understand how much time game rounds took, we logged how many seconds players spent on each sentence decision. We controlled for instances of players leaving a game open mid-annotation by applying $\min(120, t)$ to all recorded times t. We computed total time per annotation by summing the times for each sentence-level decision. We found players took longer on annotations where they ended up receiving more points, and players gradually got faster over time (Figure **??**). While one might expect longer sentences to take more time to read and make decisions on, we found no correlation between time taken

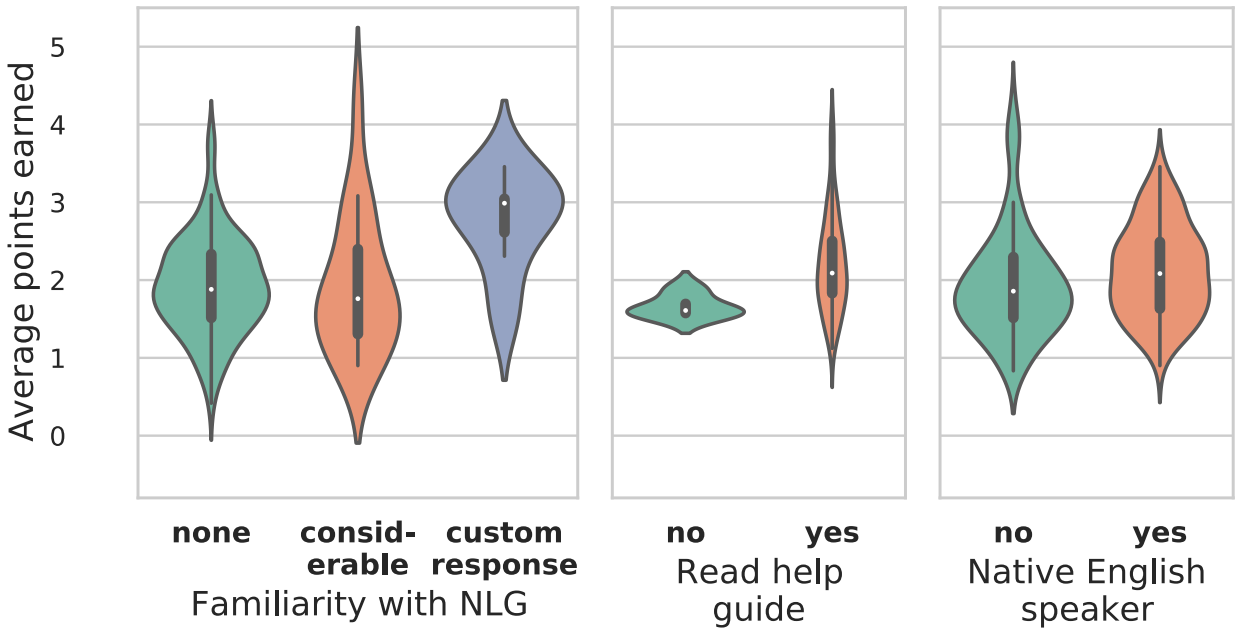


Figure 3.11: Violin plots showing results of our mandatory exit survey. A violin plot is a box plot that also provides a density estimation. Results shown are filtered to only include players who did at least 20 rounds. We see that reading the help guide, being a native English speaker, and providing a custom response for your familiarity with NLG all contribute to a higher mean score while high domain expertise does not seem have an affect (except in the case of short stories, where variance is lower for domain experts).

and length of sentence (ρ=-0.10), indicating that players take time to think about the task beyond just reading the sentence.

3.5.11 What sentence-level features could be used to detect generated text?

It has been well-studied how generated text differs in basic, measurable ways from human-written text, often due to the choice of decoding strategy. In particular, we measured how sentence length, part-of-speech distribution, and presence of named entities and novel words differed between the generated and human-written sentences in our dataset, and whether players were able to pick up on these differences. Figure **??** shows the

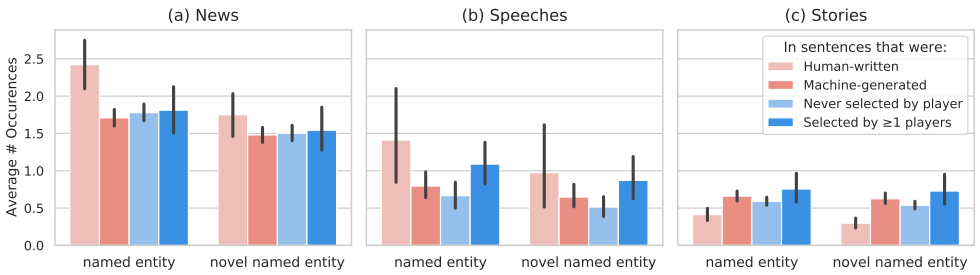


Figure 3.12: We see that human sentences tended to have a different number of named entities than generated sentences. Players picked up on the correct trend in Stories, but not in News or Speeches.

results for named entities, where novel named entities are ones which occured in the current sentence but not in any previous sentences. We found surprisingly different trends across different genres. On News and Recipes, the generated sentences tended to have fewer named entities than in human-written sentences. Annotators did not pick up on these trends, though they may have picked up on the fact that for Stories, the generated sentences tended to have slightly more named entities.

In News and Speeches, machine-generated sentences tended to be shorter than human-written ones, a trend players did not pick up on. However, for Stories, the generated sentences were on average longer than the human ones, and annotators tended to select longer sentences as the boundary. Additionally in Stories, generated sentences has on average a greater proportion of adjectives and adverbs, but annotators did not pick up on this trend.

3.5.12 Does familiarity affect detectability?

All participating players filled out an exit survey after completing their annotations. The questions on this survey are in Table **??**. Figure **??** shows some of the results. First, there was not much difference in performance between participants who reported they had

never heard of GPT-2/3 and those who reported having considerable familiarity with them. Interestingly, participants who answered "other" and wrote custom responses did end up being better at the task. Second, participants who admitted that they did not read the help guide tended to perform poorly; all the best players did read the guide. Third, there was not much difference in ability between native and non-native English speakers. The very strongest players were not native English speakers. Finally, we did not observe any correlation between self-reported familiarity with a given genre and detection skill on that genre.

3.5.13 What are the most reliable errors to look for when detecting generated text?

Each time a player specified a sentence was machine-generated, they had the option to specify why they made this decision, selecting from a set of pre-defined options (Table **??**) or else writing down a custom reason. Table **??** shows for each reason, the average number of points earned when that reason was specified. Like **clark2021all**, we see that conditioning on bad grammar is by far the least reliable way to detect generated text. In addition, we see that over 30% of all reasons given for generated text were "irrelevant," a result that stayed consistent across all models and domains. We note that the three most reliable reasons given ("common_sense," "irrelevant," and "contradicts_sentence") were also the three most common, indicating that improving these attributes will lead to the biggest improvements in generation performance. A more detailed breakdown, as well as several examples of custom reasons, can be found in Appendix **??**.

3.5.14 Discussion

In this section, we have demonstrated the viability of the boundary detection task as a framework for soliciting human evaluation of natural-language generation systems. We conducted the largest study of generated text detectability to date and, in the process, replicated many previous major results in the field, such as the improved performance of bigger models [**kaplan2020scaling**], the importance of decoding strategy selection [**ippolito-etal-2020-automatic**], and the difficulty in incentivizing annotators to improve over time [**clark2021all**]. In addition, we have provided new insights into the ways in which humans interact with partially-generated text.

In terms of ethics, work on the detectability of machine-generated text sits at an interesting balancing point. On one hand, gamifying and publicizing the detection task may help to raise the public's awareness of their susceptibility to machine-generated text, and work such as ours paves the way for future research on techniques for helping the public to improve at detection. On the other hand, we show that the detection task is a viable method for evaluating generation systems. For researchers aiming to build better generative language models, decreasing human detection ability might a very reasonable goal to optimize for. As much as our project seeks to better understand and improve human detection, our results can just as easily be used to make generative models even less detectable than they already are. Despite this drawback, we nonetheless believe it is important to study detection as a means of assessing the risks that language models pose and protecting against future harm.

One major area of improvement in our study is to develop better incentives for players to give good annotations that are unbiased by system design. Many of the students given extra credit proportional to the amount of points they scored learned they could exploit the point system by always picking one of the later sentences as the boundary. They

found that rapidly guessing Sentence 9 as the boundary on every game round was a more effective strategy for maximizing earned points per time spent than taking the time to carefully read the text in each round. One alternative system which could reduce this bias would be to show all ten sentences in the passage at once rather than show them one at a time. The player would get a certain number of tries to guess the index of the boundary sentence and would be scored based on the number of tries this takes. This would resolve the bug in our current system that some sentence positions have a high point value in expectation.

In addition to the game design aspect of the study, there are many more areas for future work to investigate. To start, our study tested only a limited subset of models and domains. It would be interesting to see if larger models, such as GPT-3 [**brownetal2020**], continue to show similar results across the variables we investigated (finetuning, topic control, decoding strategy). Another area of improvement is in the nature of the boundaries. Our study was limited in that we assumed continuations always happened on the sentence boundary, but that is not always necessarily the case. Future work should look at continuations that do not happen exactly on the boundary between sentences. While this would require major edits to the detection task, interface, and incentive structure, it would allow for a much more realistic representation of what generated text looks like in the real world if done properly.

Finally, we believe that future work should develop techniques to betetr investigate what exactly annotators are thinking when they make their decisions. Researchers need to help give annotators the language to properly explain how and why they come to select the sentences that they do, whether this be in the form of better error categorization or in free text responses. In addition, future work should seek to draw from a more diverse pool of annotators. We acknowledge that university students (many of whom have studied computer science) may not be representative of the larger population and we

hope that future work can take on a larger user study on a more diverse population and investigate how the unique backgrounds of different annotators contribute to how they detect generated text.

3.5.15 Summary of Contributions

The RoFT platofmr was introduced as a system demonstration at the 2020 Conference on Empirical Methods in Natural Language Processing [11]. The RoFT website was implemented by Arun Kirubarajan, Liam Dugan, and myself, with the assistance of Run Shi, and the mentorship or Chris Callison-Burch. The user study was designed and run, and its results analyzed, by Liam and myself.

ofsmall

Prompt: Look, nobody knows we did it.	
RS 0.5	**Standard BS**
I don't know what you're talking about.	We've got to get out of here.
What's the matter with you?	What do you mean?
I don't know what it is.	I don't think it's a good idea.
I don't think so.	I don't know what to say.
He's got to get out of here.	I don't know what's going on.
RS 1.0	**Standard BS with PDC**
I can't find it.	I don't know!
They're our ships.	I don't think so.
It's all right anyone is the right to interfere.	What do you mean?
We didn't have a plan I engineered a policy.	Why didn't you tell me?
Same time you pick us up at six and get we.	That's why we're here.
RS 1.0,top10	**RS 1.0,top10 with PDC**
I don't know what else to do.	What do you mean?
It doesn't have to be that way!	I don't think so.
We're in the air!	That's why I'm here.
I've seen a guy in his place in a it.	It's all right we.
And I'm not we any more.	We've been through this before.
NPAD0.3 BS	**Cluster5 BS**
I don't think it's a good idea.	I don't know why.
I don't know what to say.	What do you mean?
I don't know what's going on.	I don't think so.
I don't know what to do.	How do you know that?
I don't know what's going on here.	I'll tell you what.
Top3Cap BS	
We've got to get out of here.	
What do you mean?	
I don't think it's a good idea.	
I don't know what to say.	
I don't know what's going on.	

Table 3.3: Responses to an example prompt for selected methods. More examples can be seen in the appendix.

	BERT		BagOfWords		HistGLTR		Hist50Buckets		TotalProb	Human
Method	acc	AUC	acc	AUC	acc	AUC	acc	AUC	acc	acc
k40-1wordcond	0.88	0.99	0.79	0.87	0.52	0.52	0.69	0.76	0.61	0.64
p0.96-1wordcond	0.81	0.89	0.60	0.65	0.53	0.56	0.54	0.56	0.63	0.77
p1.0-1wordcond	0.79	0.92	0.59	0.62	0.53	0.55	0.54	0.55	0.65	0.71

Table 3.4: Performance (accuracy and AUC) of the fine-tuned BERT classifier and several simple baselines on detecting length-192 sequences generated with one word of priming (1worccond). Note that p1.0 refers to untruncated random sampling, where we sample from 100% of the probability mass. The last column shows human performance on the same task where accuracy with a 50% baseline is computed by randomly pairing samples from each decoding strategy with a human-written sample.

		Eval		
		top-k	nucleus	random
Train	top-k	**90.1**	57.1	43.8
	nucleus	79.1	**81.3**	78.4
	random	47.8	63.7	**81.7**
	mixed	88.7	74.2	72.2

Table 3.5: Accuracy of BERT fine-tuned discriminator when trained on samples from one strategy (rows) and evaluated on another (columns). Trained on samples with 192 tokens. The 'mixed' dataset is one containing an equal portion of samples from each strategy.

		Eval		
		top-k	nucleus	random
Train	top-k	60.9	27.9	14.5
	nucleus	49.2	51.7	48.9
	random	7.3	22.6	38.3

Table 3.6: Average probability of 'machine-generated' according to each length-192 discriminator. The expected in-domain probability is 0.5. One token of conditioning.

Truth	Raters	p1.0	k40	p0.96	Truth	Raters	p1.0	k40	p0.96
H	M	H	H	M	H	H	M	M	M
EDIT:OKAY!, I guess that'll work for now. >_ http://www.teamfortress.com/ and then go buy the game and experience some of the best online gaming I have ever played. ^__^Both girls had a really fun time and I had a GREAT time making both of these costumes. Everything was altered even a little bit(dying the pants a darker grey and painting the boots and shirts) But my piece de resistance would have to be my eldest's Medi-Gun.If you have any questions about the costumes, I would be happy to assist you!Oh and here's a video of my daughter before the costume was completed.Thanks!					Image copyright Getty Images Image caption Women mourn over the coffin of one of the victim's of Sunday's bombing in Ankara ¶Who'd be in Turkey's shoes right now? ¶Since July last year, hundreds of soldiers and civilians have been killed in terrorist attacks. Suicide bombs have torn into crowds of demonstrators and tourists. Military convoys have been targeted in the heart of the capital. ¶A long-running Kurdish insurgency, once thought to be close to resolution after years of painstaking efforts to build bridges, has erupted once more. ¶The country is awash with Syrian and other refugees. The government has been under pressure to stop them moving on into Europe and prevent would-be jihadis travelling the other way. ¶How dangerous is Turkey's unrest? ¶Tears and destruction amid PKK crackdown ¶Turkey v Islamic State v the Kurds				

Truth	Raters	p1.0	k40	p0.96	Truth	Raters	p1.0	k40	p0.96
M	M	H	-	-	M	M	-	-	H
First off, this thread has done a pretty good job of describing in detail yet another broken touchscreen. That's the difference between a smartphone and a PC with no prying eyes having to snap shots for the police to find. ¶What I would like to address is the mindset that generally surrounds Chrome OS users. To me this is analogous to saying that Apple does"hate their Windows", or that HP does"hate their Macs" as if http://twitter.com/) (and that quote is from two years ago), that anyone who covers smartphones and tablets from a "PC" perspective is just jealous. ¶Chrome OS is for browsing the web, PC processors can do stronger things in that regard, Windows is a juggernaut on those fronts. This is how I see it. Yes, it can be slow. And yes, you need a fast CPU					FOR ALABAMA, GOOD WEEKS ¶AND A TOUR OF CAIRO ¶THE ALABAMA COMMITTEE ON THE STUDY OF THE AMERICAN SECURITY AGENDA, ¶America's future has been mapped out in carved stone. Metro Atlanta's last US congressman, Bill Posey, was a inextricable integral element of the Citadel project as it became another metaphor for Atlanta's transformation from an industry backwater into the finance and information hub of the nation's capital. Meanwhile, Cobb County – Atlanta's geode of change – is home to some of the largest industrial parks in the South, a regional cultural center, a 100-year-old manufacturing town and a potent symbol of the former city's cherished Georgian past. The gentry still live there, the defunct industrial landscapes carry the names of				

Truth	Raters	p1.0	k40	p0.96	Truth	Raters	p1.0	k40	p0.96
M	H	-	-	M	M	H	-	M	-
Exidentia at Eurnari, is an upcoming Cryptopia event which is currently still in development. Be a part of the first live stream of this year's event on 15-16 January 2016! ¶Since the release of v1.22, Exidentia has received a fair amount of user feedback. This event takes place in the underwater Cryptopia they have built. During this event, you will learn about the ocean and areas around it, and be reached by a treasure hunter that helps you explore the different areas. ¶There will be six different levels in this event that you will become acquainted with: thought Polar Lava, Ocean Seared Cones and Celestine Floors, Sea Damaged Aerie Bricks, coast Puddle (congipit stopping at red water), Shaikh Swamp and Bugmite. At rotating points, you will learn how to access various types of creatures					Ever since the opening of the North American College of Art Education in 1990, the demand for art education in America has grown steadily, and in recent years we have seen the rise of students that pursue art education not in the classroom but at art academies. This year saw another 50 percent increase in the number of art academies in the United States offering courses – with an additional 10 percent of students in 2017 taking art. ¶Some major changes have occurred in recent years with regard to the art curriculum and the way students learn, and we will explore each of these in coming months as we look at the various forms of art education. There is no one-size-fits-all approach for this or any other field of study, and students who begin a course in art education may change their plans based on what they see that course, including what lessons they have completed and the resources available, to create meaningful experiences of artistic creation. ¶One important area				

Table 3.7: Some 192-token examples where at least two expert raters agreed with each other, but were not in agreement with the automatic discriminators. The first row shows examples where the ground-truth was human-written, the second shows machine-generated examples where the corresponding discriminator guessed incorrectly, and the third shows machine-generated examples where the discriminator was correct, but raters got it wrong.

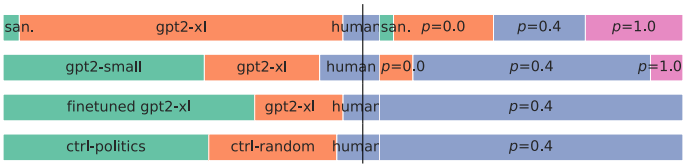
Genre	# Gens	# Annotations Raw	# Annotations Final	Avg Ann/Gen	Systems	Decoding Strategies
News	1,838	7,806	4,488	2.97		
Stories	9,864	8,007	4,614	2.53		
Recipes	7,258	17,978	7,709	2.13		
Speeches	297	8,374	4,835	16.28		

Table 3.8: Statistics on the annotation tasks (game rounds) available in our system. The discrepancies in number of annotations per dataset is partially due to the fact that players were able to choose which domain they performed annotations in.

Class	# Participants	# Annotations	Avg Ann / Part	Avg Score / Part	Avg Time (s)
Group A	141	6,527	46	1.966	5.651
Group B	102	15,119	148	2.134	6.443
Overall	241	21,646	90	2.083	6.338

Table 3.9: Statistics on the students who were invited to complete annotations on ROFT. "Avg Ann / Part" is the average number of annotations per participating student, while "Avg Score / Part" is the average score. "Avg Time" is the average time it took a participant to read one sentence. Standard error is shown.

Group	k	n	Spearman ρ
A	50	22	-0.03
A	100	13	-0.06
B	50	88	0.29
B	100	81	0.42

Table 3.10: The Spearman's rank correlation coefficient between the number of annotations performed before the current annotation and the score on the current annotation, for all n players who have performed k or more annotations. Players in Group B, who were given extra instruction and incentives, improved over time while those in Group A did not.

Reason	n	Mean Score
common_sense	2,432	2.566 ± 0.086
irrelevant	4,259	2.530 ± 0.064
contradicts_sentence	1,606	2.527 ± 0.105
contradicts_knowledge	1,411	2.262 ± 0.111
coreference	542	2.249 ± 0.176
repetition	728	2.128 ± 0.154
other	75	2.040 ± 0.483
generic	1,546	1.920 ± 0.101
grammar	1,539	1.780 ± 0.105

Table 3.11: The number of times each reason was given for text being machine-generated, and the mean score over those annotations. We see that when players select reasons like "grammar" or "generic," they are much less likely to be correct than when selecting "common_sense" or "irrelevant."

4 MEMORIZATION

4.1 MOTIVATION

The performance of neural language models has continuously improved as these models have grown from millions to trillions of parameters [**fedus2021switch**], with their training sets similarly growing from millions to trillions of tokens. However, larger models trained on increasingly large and minimally curated datasets are at increased risk of memorizing their training data. Carlini et al. [**carlini2020extracting**] found that while most instances of memorization are innocuous, such as news articles or religious text, models are also capable of memorizing things like contact information or the names of real individuals (referenced outside of news contexts).

In this chapter, we quantify the properties which raise the risk of memorization and then describe one actionable step–thorough dataset deduplication–which can be employed to diminish the risk of memorization.

It is important to quantify factors that lead to increased memorization of a model's training set. Indeed, recent work has shown that *training data extraction attacks* are a practical threat for current language models [**carlini2020extracting**]; an adversary interacting with a pretrained model can extract individual sequences that were used to train the model.

While current attacks are effective, they only represent a lower bound on how much memorization occurs in existing models. For example, by querying the GPT-2 language

model, **carlini2020extracting** (manually) identified just 600 memorized training examples out of a 40GB training dataset. This attack establishes a (loose) lower bound that at least 0.00000015% of the dataset is memorized. In contrast, we are able to show that the 6 billion parameter GPT-J model [**gpt-neo**, **gpt-j**] memorized at least 1% of its training dataset: The Pile ([**gao2020pile**]).

In addition to these loose estimates of models' memorization capabilities, there is a limited understanding of how memorization varies across different neural language models and datasets of different scales. Prior studies of memorization in language models either focus on models or datasets of a fixed size [**carlini2019secret**, **zhang2021counterfactual**, **thakkar2020understanding**] or identify a narrow memorization-versus-scale relationship [**carlini2020extracting**, **2021dedup**]. **mccoy2021raven** broadly study the extent to which language models memorize, but their focus is on how to avoid the problem and ensure novelty of model outputs, rather than on studying model risk through identifying maximum memorization.

This paper addresses both of the above open questions by comprehensively quantifying memorization across three families of neural language models and their associated datasets. We leverage access to each model's original training set to provide order-of-magnitude more precise bounds on the amount of extractable data than in prior works.

To construct a set of prompts from the model's training set, we feed varying-length prefixes of the training data back into the trained model, and verify whether the model has the ability to complete the rest of the example verbatim. This allows us to measure memorization across models, datasets, and prompts of varying sizes. We identify three properties that significantly impact memorization:

1. **Model scale:** Within a model family, larger models memorize 2-5$\times$ more data than smaller models.

2. **Data duplication:** Examples repeated more often are more likely to be extractable.
3. **Context:** It is orders of magnitude easier to extract sequences when given a longer surrounding context.

Our analysis suggests that future research on neural language modeling will need to take steps to prevent future (larger) models from memorizing their training datasets.

4.2 MOTIVATION

Machine-generated text is most undetectable when it looks exactly like its training data. Given that neural language models are trained to maximize the likelihood of their training data, what's stopping them from simply memorizing and regurgitating exact text sequences they were trained on? This sort of memorization is directly harmful if it breaches expectations of privacy or content ownership from those whose data is included in the train set. It also reduces generalizability if models are biased toward examples that are not representative of the underlying distribution of natural language. In our recent paper [4], I show that large numbers of near-duplicate examples in the training data for neural language models significantly increases their tendency to memorize (Figure 4). Deduplication of the training data results in more efficient training (since dataset size is reduced) without harming model performance, and providing improved generalization to out-of-domain text. In ongoing work, my team is measuring the scaling laws of memorization, showing that memorization frequency increases logarithmically with model size and the number of times a training example is repeated.

4.3 DEDUPLICATING TRAINING DATA REDUCES MEMORIZATION

4.3.1 Introduction

A key factor behind the recent progress in natural language processing is the development of large-scale text corpora used to train increasingly large language models. These datasets have grown from single gigabytes to as much as a terabyte over the past few years [**chelba2013one**, **xue2020mt5**, **graff2003english**, 4]. Because it is so expensive to perform manual review and curation on massive datasets, they tend to suffer in quality compared to their smaller predecessors. This has implications far beyond metrics like perplexity and validation loss, as learned models reflect the biases present in their training data [**bender2021stochastic**, **wallace2019universal**, **sheng2020towards**]. As a result, quantitatively and qualitatively understanding these datasets is therefore a research challenge in its own right [**dodge2021documenting**].

We show that one particular source of bias, duplicated training examples, is pervasive: 10% of the sequences in several common NLP datasets are repeated multiple times. While naive deduplication is straightforward (and the datasets we consider already perform some naive form of deduplication), performing thorough deduplication at scale is both computationally challenging and requires sophisticated techniques.

The simplest technique to find duplicate examples would be to perform exact string matching between all example pairs, but we show this is insufficient since the web containing many docments which are near-duplicates of eacho ther. This, we introduce two complementary, scalable methods for performing deduplication on documnets which have substantial overlap but may not be identical.

- *Exact* substring matching identifies strings that are repeated verbatim in the train set multiple times. This allows us to identify cases where only part of a training

example is duplicated (§**??**). Using a suffix array [**manber1993suffix**], we are able to remove duplicate substrings from the dataset if they occur verbatim in more than one example.

- *Approximate* full document matching uses MinHash [**broder1997resemblance**], an efficient algorithm for estimating the n-gram similarity between all pairs of examples in a corpus, to remove entire examples from the dataset if they have high n-gram overlap with any other example (§**??**).

We identify four distinct advantages to training on datasets that have been thoroughly deduplicated.

1. Over 1% of tokens emitted unprompted from a model trained on standard datasets (e.g., C4) are part of a memorized sequence (See §**??**)—even though the 1.5 billion parameter model is much smaller than the 350GB dataset it was trained on. By deduplicating the training dataset we reduce the rate of emitting memorized training data by a factor of $10\times$.

2. Train-test overlap is common in non-deduplicated datasets. For example, we find *a 61-word sequence*[14] in C4 [39] that is repeated 61,036 times verbatim in the training dataset and 61 times in the validation set (0.02% of the samples in each dataset). This train-test set overlap not only causes researchers to over-estimate model accuracy, but also biases model selection towards models and hyperparameters that intentionally overfit their training datasets.

3. Training models on deduplicated datasets is more efficient. Processing a dataset with our framework requires a CPU-only linear-time algorithm. And so because

14 "by combining fantastic ideas, interesting arrangements, and follow the current trends in the field of that make you more inspired and give artistic touches. We'd be honored if you can apply some or all of these design in your wedding. believe me, brilliant ideas would be perfect if it can be applied in real and make the people around you amazed!"

these datasets are up to 19% smaller, even including the deduplication runtime itself, training on deduplicated datasets directly reduces the training cost in terms of time, dollar, and the environment [**bender2021stochastic**, **strubell2019energy**, **patterson2021carbon**].

4. Deduplicating training data does not hurt perplexity: models trained on deduplicated datasets have no worse perplexity compared to baseline models trained on the original datasets. In some cases deduplication reduces perplexity by up to 10%. Further, because recent LMs are typically limited to training for just a few epochs [38, 39], by training on higher quality data the models can reach higher accuracy faster.

To summarize, data duplication offers significant advantages and no observed disadvantages. In the remainder of this paper we present our text deduplication framework in §**??**, and study the extent of duplicate content in common NLP datasets (e.g., C4, Wiki-40B, and LM1B) in §**??**. We then examine the impact of deduplication on test perplexity (§**??**) and on the frequency of emitting memorized content (§**??**). Finally, we analyze to what extent perplexity on existing, released models are skewed as a result of overlap between the train and test/validation splits (§**??**).

4.3.2 Related Work

Large Language Model Datasets

While we believe our results are independent of model architecture, we perform our analysis on Transformer-based decoder-only language models [44] trained for open-ended text generation. These current state-of-the-art models are trained on internet text. For example, the GPT-2 family of models Radford et al. [38] is trained on WebText, a dataset of web documents highly ranked on Reddit—however this dataset was not made available

publicly. A common dataset starting point is CommonCrawl, an index of public webpages. Among the models trained on CommonCrawl include GPT-3 [4] with the addition of book datasets, GROVER [**zellers2019defending**] on a restricted subset filtered to news domains called RealNews, and T5 [39] on a cleaned version of common crawl called C4. Other models are trained on more curated Internet sources—for example **guo2020wiki40b** used high quality processed Wikipedia text from 40 different languages to train monolingual 141.4M parameter language models. Non-English models necessarily use different datasets; **zeng2021pangualpha** for instance introduced PANGU-α, a family of models with up to 200B parameters that were trained on a non-public corpus of cleaned and filtered Chinese-language documents from CommonCrawl and other sources. Since many of these datasets are not public, we deduplicate three that are: Wiki-40B, C4, and RealNews–as well as the One Billion Word Language Model Benchmark [**chelba2013one**], a smaller dataset commonly used for evaluation.

Others have observed that popular datasets contain problematic duplicate content. Bandy et al. [**bandy2021addressing**] observe that the Book Corpus [**zhu2015aligning**], which was used to train popular models such as BERT, has a substantial amount of exact-duplicate documents according to. Allamanis et al. **allamanis2019adverse** show that duplicate examples in code datasets cause worsened performance on code understanding tasks.

Contamination of Downstream Tasks

When models are trained on datasets constructed by crawling the Internet, it is possible the model will train on the test set of downstream target tasks. For example, Radford et al. [38, §4] performed a post-hoc analysis to identify 8-gram overlaps between GPT-2's training set and datasets used for evaluation, and **Dodge2021-lb** analyzed C4 and found that up to 14.4% of test examples for various standard tasks were found verbatim (normalizing

for capitalization and punctuation) in the dataset. A more proactive approach removes contaminated data. **trinh2018simple** removed documents from their CommonCrawl-based train set that overlapped substantially with the commonsense reasoning used for evaluation. And GPT-3 [4, §5] did the reverse and removed downstream evaluation examples from their training data by conservatively filtering out any train set examples with a 13-gram overlap with any evaluation example. Up to 90% of tasks were flagged as potentially contaminated.

In our research, we do not focus on the impact of duplicate text in pretrained models on downstream benchmark tasks; instead we address how duplicate text in the LM training and validation sets impacts model perplexity and the extent to which generated text included memorized content.

Memorizing training data.

The privacy risks of data memorization, for example the ability to extract sensitive data such as valid phone numbers and IRC usernames, are highlighted by **carlini2020extracting**. While their paper finds 604 samples that GPT-2 emitted from its training set, we show that *over* 1% of the data most models emit is memorized training data. In computer vision, memorization of training data has been studied from various angles for both discriminative and generative models [**arpit2017closer**, **8953411**, **feldman2020neural**, **stephenson2021geometry**]

4.3.3 Datasets Considered

We analyze the presence of duplicate text in four datasets of varying sizes that have been used for training natural language generation systems, producing general-purpose pre-trained models, and for language model benchmarking. While this paper restricts

itself to English datasets, we expect that non-English datasets suffer from similar issues and could likewise benefit from de-duplication.

Wikipedia (Wiki-40B)

consists of multi-lingual cleaned Wikipedia text [**guo2020wiki40b**]. We take the English portion, which contains 2.9M Wikipedia pages with an average length of 768 BPE tokens. The dataset creators do not indicate any deduplication was performed aside from removing redirect-pages (e.g., "sunflower" to "Helianthus").

One-Billion Word benchmark (LM1B)

contains 30M sentences of news commentary [**chelba2013one**]. Unlike the other datasets we analyze, LM1B's examples are one sentence long rather than multi-sentence documents. The average example length is 32 BPE tokens. While this dataset is extremely standard for benchmarking language models, Radford et al. [38, Sec 4] note it has 13.2% overlap of the test set with the train set.

Colossal Cleaned Common Crawl (C4)

is made up of 360M web documents, with an average length of 486 BPE tokens [39]. C4 was introduced as a pre-training dataset for T5, a set of encoder-decoder models which have been widely used in fine-tuned downstream tasks. The dataset was previously deduplicated in a more sophisticated process than the prior two datasets. Each paragraph was hashed and paragraphs resulting in hash collisions were removed. This was followed by a pass that removed placeholder text, code, and prohibited words. See **dodge2021documenting** for a detailed breakdown of the source text in C4.

RealNews

is a subset of the Common Crawl consisting of articles from news domains [**zellers2019defending**]. It contains 31M documents with average length 793 BPE tokens. RealNews was deduplicated by inserting a hash of the first 100 characters of each document into a bloom filter [**bloom1970space**] and then excluding any document which resulted in a hash collision. Like C4, examples with duplicate URLs were excluded.

4.3.4 Method for Exact Substring Duplication

We consider a dataset $D = \{x_i\}_{i=1}^{N}$ as a collection of *examples* x_i. Each of these examples is itself a sequence of *tokens*: $x_i = \left[x_i^1, x_i^2, \cdots, x_i^{s_i}\right]$.

Due to the diversity of possibilities in human language, it is rare for the same idea to be expressed identically in multiple documents unless one expression is derived from the other, or both are quoting from a shared source. This observation motivates deduplicating exact substrings. We call our approach ExactSubstr. When two examples x_i and x_j share a sufficiently long substring (that is, a substring for which $x_i^{a..a+k} = x_j^{b..b+k}$), that substring is removed from one of them.

Suffix Arrays

This exact-substring-matching criterion, while conceptually simple, is computationally prohibitive with naive (quadratic) all-pair matching. To improve the efficiency, we concatenate all the examples of the entire dataset D into a giant sequence $\mathcal{S}$, and construct a Suffix Array $\mathcal{A}$ of $\mathcal{S}$. A suffix array [**manber1993suffix**] is a representation of a suffix tree [**weiner1973linear**] that can be constructed in linear time in $\|\mathcal{S}\|$ [**karkkainen2003simple**] and enables efficient computation of many substring queries;

in particular, they allow us to identify duplicated training examples in linear time. Suffix arrays have the advantage over suffix trees in that they are 10–100× more memory efficient [**manber1993suffix**], requiring just 8 bytes per input token, though they are asymptotically less efficient for some query types. They have been used widely in NLP, such as for efficient TF-IDF computation [**yamamoto2001using**] and document clustering [**hung2007new**].

The suffix array $\mathcal{A}$ for a sequence $\mathcal{S}$ is a lexicographically-ordered list of all suffixes contained in the sequence. Formally,

$$\mathcal{A}(\mathcal{S}) = \arg\,\text{sort}\,\text{all\_suffixes}(\mathcal{S})$$

For example, the suffixes of the sequence "banana" are ("banana", "anana", "nana" "ana", "na", "a") and so the suffix array is the sequence (6 4 2 1 5 3). In practice, we construct $\mathcal{S}$ from the BPE tokenization of the text (§**??**).

Substring matching

After constructing $\mathcal{A}$, it is straightforward to identify duplicated training examples. Suppose that the sequence s was repeated exactly twice in the training dataset $\mathcal{S}$ at positions i and j, that is, $\mathcal{S}_{i..i+|s|} = \mathcal{S}_{j..j+|s|}$. Then the indices i, j will occur adjacent to each other in the suffix array $\mathcal{A}$.

Finding all repeated sequences is thus a matter of linearly scanning the suffix array from beginning to end and looking for sequences $\mathcal{A}_i, \mathcal{A}_{i+1}$ that share a common prefix of at least some threshold length. Any satisfying sequences are recorded.

Setting a Threshold of Duplicates

One important question is how long a substring match must be before we ought to count it as a duplicate. In Figure **??**, we plot the frequency of substring matches within the four datasets we will consider. For each substring of length k, we compute the probability that there exists another sequence of length k identical to this one; formally:

$$m(k) = \Pr_{i \in [N]} \left[\exists j \neq i : \mathcal{S}_{i..i+k} = \mathcal{S}_{j..j+k} \right].$$

We choose 50 tokens as the threshold to be conservative: the "bend in the knee" occurs at 10 tokens, and manual inspection of length-25 matches found no false positives. We then doubled this value to have an exceptionally large margin for error.

Implementation

PARALLEL LINEAR TIME CONSTRUCTION. We build a parallelized linear time suffix array algorithm. As a building block, we make black-box use of the SA-IS algorithm for constructing a suffix array in linear time **nong2009linear**, **ko2003space**. Unfortunately, this algorithm is not easily parallelized directly, so we introduce a simple divide and conquer approach to parallelizing the array construction.

We build our implementation in Rust and extend an existing suffix array library[15] with three modification. The first two are straightforward implementation differences: we modify the code to allow datasets larger than 4GB, and we remove the requirement that strings parse as valid UTF-8 sequences in favor of raw byte sequences. Our third change is more significant: we re-implement the algorithm so that we can stream the suffix array itself off disk.

15 https://github.com/BurntSushi/suffix

PARALLEL PARTIAL SUFFIX ARRAY CONSTRUCTION. Our divide and conquer suffix array construction algorithm starts by partitioning the dataset into K different "splits" with SA-IS run over independently on each split in parallel. This algorithm still requires $O(N)$ work but runs in $O(N/K)$ wall-clock time. This gives us N separate suffix arrays $\mathcal{A}^i$.

Given two suffix arrays A_1 and A_2 for two sequences S_1 and S_2 it's not completely trivial to construct a single suffix array A for $S = S_1 \mid\mid S_2$ because of the boundary conditions. Instead, we don't build the data $S = S_1 \mid\mid S_2$ but rather let $S_1' = S_1 \mid\mid S_2[uptoK]$ for some K greater than the longest substring match. Then we build the arrays on S_1' and S_2. To merge the arrays together we can remove the items from the first array after index $|S_1|$ and merge-sort insert them into the second.

PARALLEL MERGE OF PARTIAL SUFFIX ARRAYS. We now merge these separate arrays together into a single suffix array $\mathcal{A}$, Consider the simpler case of two partial suffix arrays B and C that we would like to merge together. We can achieve this by letting $i = 0$ index B and $j = 0$ index C. Each iteration of the algorithm then pushes B_i into $\mathcal{A}$ if $S_{B_{i..}} < S_{C_i}$ and C_i otherwise, repeating until $i = |B| - 1$ and $j = |C| - 1$. To generalize to K splits, we need only replace the single comparison above with a min-heap requiring $O(\log K) \ll 10$ work on each iteration.

Observe that in the general case this algorithm is $O(Nm\log(K))$ where N is the length of the dataset, m is the average length of a prefix match, and K is the number of splits. It is therefore incorrect to call this algorithm linear time in the general case, for ours it is. Because the length of the longest match is bounded above by the length of the longest sequence, as long as the size of the dataset is independent of the length of the longest sequence in the dataset, this algorithm remains efficient.

Again, we can parallelize this operation among L simultaneous jobs (in practice we set $K = L$ as the number of threads on our machine). In the $K = 2$ case, job l processes

$i \in [jN/L, (j+1)N/L]$, choosing the bounds of j by binary searching into C so that $S_{B_i} < S_{C_j} < S_{B_{j+1}}$. The case where $K > 2$ is identical except that we repeat this over all K partial suffix arrays.

Computational Analysis

We run our algorithm on a single VM on the cloud with 96 cores and 768GB of memory. Our algorithm is efficient, for example processing the Wiki-40B training set (3 million examples containing 4GB of text) in 2.3 minutes wall-clock time (2.1 CPU-hours of work). The 350GB C4 dataset takes under 12 hours (wall-clock) to build a suffix array; although we are still memory constrained and so this corresponds to ~ 1000 CPU-hours. Once the suffix array has been constructed, it takes under an hour to deduplicate the C4 dataset.

Note that this algorithm still requires that the dataset itself fits in memory (so that we can efficiently index in arbitrary positions), but we do not need to fit the entire suffix array into memory. This is fortunate since our suffix array requires an $8\times$ space overhead. For example, the suffix array for the 350GB C4 is 1.5TB.

Compared to the cost of training a language model on this dataset, the additional work required to deduplicate the training dataset is negligible.

4.3.5 Method for Approximate Matching with MinHash

Overview

We also perform *approximate* deduplication based on matching entire examples. This method, which we call NearDup, is a good complement to the *exact* substring matching, especially for web crawl text, as it handles the very common case of documents being identical except for interspersed templated fields (such as the last row of Table **??**).

Table 4.1: Qualitative examples of near-duplicates identified by NearDup from each dataset. The similarlity between documents is highlighted. Note the small interspersed differences that make exact duplicate matching less effective. Examples ending with "[...]" have been truncated for brevity. More data available in Appendix.

Dataset	Example	Near-Duplicate Example
Wiki-40B	\n_START_ARTICLE_\nHum Award for Most Impactful Character \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]	\n_START_ARTICLE_\nHum Award for Best Actor in a Negative Role \n_START_SECTION_\nWinners and nominees\n_START_PARAGRAPH_\nIn the list below, winners are listed first in the colored row, followed by the other nominees. [...]
LM1B	I left for California in 1979 and tracked Cleveland 's changes on trips back to visit my sisters .	I left for California in 1979 , and tracked Cleveland 's changes on trips back to visit my sisters .
C4	Affordable and convenient holiday flights take off from your departure country, "Canada". From May 2019 to October 2019, Condor flights to your dream destination will be roughly 6 a week! Book your Halifax (YHZ) - Basel (BSL) flight now, and look forward to your "Switzerland" destination!	Affordable and convenient holiday flights take off from your departure country, "USA". From April 2019 to October 2019, Condor flights to your dream destination will be roughly 7 a week! Book your Maui Kahului (OGG) - Dubrovnik (DBV) flight now, and look forward to your "Croatia" destination!

MinHash [**broder1997resemblance**] is an approximate matching algorithm widely used in large-scale deduplication tasks [**versley2012not**, **GABRIEL201863**, **gyawali2020deduplication**], including to deduplicate the training set for a large Chinese-language LM [**zeng2021pangualpha**]. Given two documents x_i and x_j, the main idea is to represent each document by its respective set of n-grams d_i and d_j. We can then use hash functions to approximate the *Jaccard Index* [**jaccard1912distribution**]:

$$\text{Jaccard}(d_i, d_j) = {}^{|d_i \cap d_j|}/_{|d_i \cup d_j|} \tag{4.1}$$

If the Jaccard Index between d_i and d_j is sufficiently high, it is likely that documents are approximate matches of each other. To efficiently approximate the Jaccard index, MinHash constructs document signatures by sorting each of the n-grams via a hash function, and

then keeping only the k smallest hashed n-grams. There are multiple ways to construct estimators of the Jaccard index from these kinds of signatures [**cohen2016min**].

In our implementation, we use 5-grams and a signature of size 9,000. The probability that two documents are considered a potential match is

$$\Pr(d_i, d_j | \operatorname{Jaccard}(d_i, d_j) = s_{i,j}) = 1 - (1 - s_{i,j}^b)^r \tag{4.2}$$

where $b = 20$ and $r = 450$ are user-settable parameters to control the strength of the filter.

For each pair of documents identified as a potential match, more computationally expensive similarity metrics can be employed as a subsequent filtering step. In particular, we identify two documents as duplicates if they are matched by the MinHash algorithm and their *edit similarity* is greater than 0.8. The edit similarity between token sequences x_i and x_j is defined as:

$$\operatorname{EditSim}(x_i, x_j) = 1 - \frac{\operatorname{EditDistance}(x_i, x_j)}{\max(|x_i|, |x_j|)} \tag{4.3}$$

To build clusters of similar documents, we construct a graph that has an edge between two documents if they are considered a match. Then, we use the method introduced in **lacki2018connected** to identify connected components.

Implementation Details

For our MinHash based deduplication method, documents are first space tokenized, then each consecutive 5-gram is hashed using tabulation hashing. The set of these hashes is the signature for the document. For each element in a document's signature, the element is hashed using k other hash functions. The minimum hashed element for each of the k hash functions is stored. These minimum hashes are then partitioned into r buckets, with b hashes per bucket. These b hashes are augmented into a single value, then if two

documents have the same value in at least one bucket, they'll be marked as a potential match. The probability that two documents are considered a potential match is equal to

$$\Pr(d_i, d_j | \text{Jaccard}(d_i, d_j) = s_{i,j}) = 1 - (1 - s_{i,j}^b)^r \tag{4.4}$$

where $s_{i,j}$ is the Jaccard index between the two documents. For document pairs that were identified as potential matches, we computed their actual Jaccard index, and if that was above 0.8, we computed their edit similarity. Document pairs with edit similarity higher than 0.8 were identified as duplicates. After some experimentation, we chose to use $b = 20$, and $r = 450$, so $k = 9{,}000$, so as to make sure a collision at the desired Jaccard index threshold of 0.8 had a high probability of occurring

We also tested an alternative configuration—filtering to document pairs with Jaccard index of at least 0.9 and edit similarity of at least 0.9. In this case, we used $b = 20$, $r = 40$, and $k = 800$. Figure **??** shows the histogram of Jaccard similarities and edit similarities for all document pairs which collided in min-hash space, for our chosen configuration (blue) and for the alternative configuration (orange). This allows us verify if the threshold chosen has few comparisons around the chosen threshold, then we've likely captured the majority of actual near duplicates above that threshold. To verify that yourself, look at the left hand tails of the distributions. Since both 0.8 and 0.9 begin to vanish at the same point (in spite of the fact that the two thresholds are optimized for accuracy around different thresholds), we feel comfortable saying that we're capturing the majority of actual near duplicates.

Computational Analysis

Let N be the number of documents and T be the maximal number of tokens in a document. Edit similarity has a worst case complexity of T^2, so the worst case complexity is

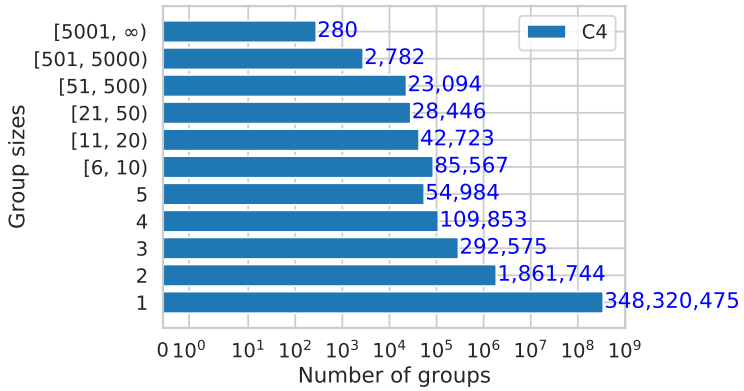


Figure 4.1: The distribution of near-duplicate cluster sizes from running NearDup on C4.

$$O(N + bk^2T^2N) = O(N) \tag{4.5}$$

since b, k, and T are all $\ll N$. The left term is the complexity of grouping by the signatures, and the right represents the pathological worst case of all documents falling into the same B buckets.

The highly distributed NearDup implementation we employed is one used for large-scale production tasks at Google. On the English C4 dataset, the algorithm consumed approximately 41.5 kWh of energy. Note that our choices of k and b were designed to produce very high recall, and with different parameters, the algorithm could be made much more energy efficient while producing similar results.

4.3.6 Results

We deduplicate each of the four datasets with both of our two techniques. When text was duplicated across multiple data splits, we prioritized keeping a copy in the test or validation set and removing it from the train set.

Amount of Text Removed

With NearDup, we found that the web-scrape datasets contain between 3.04% (on C4) to 13.63% (on RealNews) near duplicates (Table **??**). Near-duplicate text is much less common in Wiki-40B, forming only 0.39% of the train set.[16] In C4, the majority (1.8M) of near-duplicate clusters consisted of just a single pair of examples that matched against each other, but there were 280 clusters with over 5,000 examples in them (Figure **??**), including one cluster of size 250,933.

On average with ExactSubstr, we remove more total content than with NearDup (despite ExactSubstr not removing any examples outright)—for example removing 7.18% of the tokens in C4. The exception is LM1B, where ExactSubstr removes 8× less data than NearDup. On investigation, we find this is due to the fact that LM1B documents are significantly shorter: 90% of all documents are under 50 tokens, and so are not even candidates for potential matches even if the entire sequence matched verbatim. We find that both NearDup and ExactSubstr remove similar content—77% of the training examples that NearDup removes from C4 have at least one verbatim length-50 match found by ExactSubstr.

4.3.7 Properties of Duplicated Text

While the authors of both RealNews and C4 explicitly attempted deduplication during dataset construction, the methods were insufficient to capture the more subtle types of duplicate text commonly found on the internet. In C4 and Wiki-40B, we qualitatively observe that much of the text identified as near-duplicated is computer-generated. The text is identical except for the names of places, businesses, products, dates, and so on. Because

16 Most duplicates we saw were automatically generated pages, such as the outcomes of sports games. This shows the strength of manual curation for creating high-quality datasets.

Table 4.2: The fraction of tokens (note Table **??** reports the fraction of *examples*) identified by ExactSubstr as part of an exact duplicate 50-token substring.

	% train examples with		% valid with
	dup in train	dup in valid	dup in train
C4	3.04%	1.59%	4.60%
RealNews	13.63%	1.25%	14.35%
LM1B	4.86%	0.07%	4.92%
Wiki40B	0.39%	0.26%	0.72%

Table 4.3: The fraction of examples identified by NearDup as near-duplicates.

	% train tokens with		% valid with
	dup in train	dup in valid	dup in train
C4	7.18%	0.75 %	1.38 %
RealNews	19.4 %	2.61 %	3.37 %
LM1B	0.76%	0.016%	0.019%
Wiki40B	2.76%	0.52 %	0.67 %

Table 4.4: On the left, we show the URLs that had the greatest proportion of examples marked as near-duplicates by NearDup(filtered to URLs which occurred at least 10 times). On the right, we show the 20 most frequent URLs in C4 for which all examples were marked as near-duplicates by NearDup.

RealNews Url	# Total	Frac Dups	C4 Url	# Total	Frac Dup
medicalnewstoday.com.	12	1.00	hairtechkearney.com	4883	
dodbuzz.com	301	0.99	keywordsking.com	1786	
undertheradar.military.com	187	0.97	sydneysitalianfruitshops.online	1178	
q.usatoday.com	33	0.94	moewiki.usamimi.info	1001	
ad-test.thirdage.com	354	0.94	swarovskijewelryoutlet.org	984	
amp.nymag.com	15	0.93	forzadurto.org	980	
citizenwire.com	1022	0.93	producerati.com	971	
paycheck-chronicles.military.com	363	0.92	sourceryforge.org	908	
product-reviews.net	73403	0.92	heavenz-kitchen.com	876	
kitup.military.com	196	0.92	little-eclipse.com	822	
gcaptain.com	33903	0.92	walops.com	819	
dev.screenrant.com	70	0.91	16thstlaunderland.com	713	
live.swissinfo.ch	66	0.91	theroyalstarinfo.com	696	
news.theepochtimes.com	82	0.87	code4kt.com	684	
opinion.toledoblade.com	986	0.87	nflfalconsjerseys.us	682	
cdn.moneytalksnews.com	121	0.86	quiltingbeeshop.com	676	
amp.fox23.com	14	0.86	ulifeinsurancemiami.com	675	
sales.rollingstone.com	20	0.85	wowkeyword.com	673	
ftp.screenrant.com	20	0.85	taspetro.com	671	

these examples frequently differ by just a few words at a time, deduplication strategies relying on exact string matching would fail to identify a match. Example duplicate pairs from each dataset can be found in Table **??**. Table **??** shows the URLs had the largest proportion of examples identified by NearDup as near-duplicates. For C4, these tend to be websites that sell many similar products and thus have a large amount of templated text. For RealNews, content aggregators seem especially common.

For RealNews and LM1B, derived from news sites, we observe that many near-duplicates occur because the same news article appears on multiple news sites with slightly different formatting. For example, in LM1B, there is one example that starts "*MINEOLA , N.Y. - New York officials say* [...]" and another that starts "*(AP) - New York officials say* [...]". The two examples are otherwise identical.

Train / Test Set Leakage

Both deduplication methods identify overlap between the train set and the validation set (Table **??**). For example, 4.6% of the C4 validation set and 14.4% of the RealNews validation set examples had an approximate duplicate in their respective training sets. Such duplication is problematic since it could cause evaluation metrics to be unfairly inflated for models that are better at memorizing their train sets. We evaluate the effect of this leakage on publicly released models in Section **??**.

Impact on Trained Models

. We trained 1.5B parameter "XL", decoder-only, Transformer-based language models similar to GPT-2, on C4-Original, C4-NearDup, and C4-ExactSubstr, respectively. We use the T5 codebase and model architecture from Raffel et al. [39], and each model was trained for about two epochs on its respective dataset. To better understand the amount of variance in the perplexities of trained models, we also trained three different random

seeds of the 110M parameter “base" model for each of the above three datasets—for a total of nine base-sized models.

For all experiments, we used a Byte Pair Encoding (BPE) vocabulary trained on C4-NearDup with a budget of 50K tokens, which resulted in a vocabulary the same size as GPT-2's. We trained with a maximum sequence length of 512 tokens (for longer documents, we randomly extracted subsequences of this length.) Each model was trained for about two epochs. Since both C4-Original and C4-ExactSubstr contain approximately 365M examples, we performed 152K steps with a batch size of 4800 (or approximately 2 epochs). C4-NearDup contains approximately 350M examples, we performed 146K steps (or approximately 2 epochs). On a 128-core TPU v3 pod slice, XL models trained on C4-Original and C4-ExactSubstr took approximately 131 hours (5.5 days) to train, while the XL model trained on C4-NearDup took approximately 126 hours to train. Like T5, models were trained with the Adafactor optimizer [**shazeer2018adafactor**]. A constant learning rate of 0.01 was used for the base models and 0.001 for the XL models.

The 1.5B parameter XL models had 24 layers, each with 32 attention heads. The model embedding size was 2,048, the feed forward layers had a hidden size of 5,120, and the key/value dimension size for the attention heads 64. The 110M parameter base models had 12 layers, each with 12 attention heads. The model embedding size was 768, the feed forward layers had a hidden size of 2,048, and the key/value dimension size for the attention heads 64.

MODEL PERPLEXITY We computed the perplexity of our trained models on the validation sets of LM1B and Wiki-40B, and on subsets of the C4 validation set (Figure **??**). For the base size, we observe that all models have similar perplexity on the original C4 validation set and on validation set examples that were identified as unique (no near-duplicate in either train or validation). However, both models trained on deduplicated data have

significantly higher perplexity on validation set examples that have duplicates in the training set than the model trained on the original C4. ExactSubstr-deduplicated results in higher perplexity than NearDup-deduplicated. These trends holds true for the XL sized model as well. While this may suggest ExactSubstr duplication results in models least overfit on the train set, note that both of these techniques have used separate duplicate thresholds and a different choice of thresholds could change the results.

When evaluating on the validation sets of LM1B and Wiki-40B, we found that models trained on NearDup-deduplicated C4 consistently achieved lowest perplexity. ExactSubstr deduplication decreases perplexity of the XL model by almost 3 points perplexity on Wiki-40B which is much larger than the variation of about 1 point perplexity we observed in the base models. This is despite seeing fewer tokens of training data overall.

Lastly, we note all our XL models achieved <35 perplexity on LM1B, which is less than the 42.16 perplexity reported for the 1.5B GPT-2 using a vocabulary the same size as ours.

Impact on Generated Text

Data duplication has the effect of biasing the trained LM towards particular types of examples. This can contribute to a lower diversity of generations, and increased likelihood that the generated content is copied from the training data [**carlini2020extracting**]. For our generation experiments, we use top-k random sampling with $k = 50$ and experiment with prompted and unprompted generation.

We first evaluate memorization tendencies in the case where the model is asked to generate text without any prompt sequence. We generate 100,000 samples, each up to 512 tokens in length. For each generated token, we say the token is memorized if it is part of a 50-token substring that is exactly contained in the training data. On XL-Original, over 1% of the generated tokens belong to memorized sub-sequences (see Table **??**). This is $\sim 10\times$

Table 4.5: When generating 100k sequences with no prompting, over 1% of the tokens emitted from a model trained on the original dataset are part of a 50-token long sequence copied directly from the training dataset. This drops to 0.1% for the deduplicated datasets.

Model	1 Epoch	2 Epochs
XL-Original	1.926%	1.571%
XL-NearDup	0.189%	0.264%
XL-ExactSubstr	0.138%	0.168%

more memorization than XL-ExactSubstr or XL-NearDup. Some example subsequences that were copied verbatim from the train set can be found in Table **??**.

In most real use cases, language model generation is controlled by providing a prompt for the model to continue. We experiment with four possible prompt sources: training examples identified by ExactSubstr as having near-duplicates in the train set (train dup), training examples identified as unique (train unique), validation set examples with a near-duplicate in the train set (valid in train), and validation set examples identified as unique across all splits (valid unique). We select the first 32 tokens of each example as the prompt, which means we can evaluate the fraction of generations which are near-duplicates with the ground-truth continuation for the prompt. Figure **??** shows the proportion of generations which meet this requirement, while Figure **??** shows the distribution in edit similarities between the generations and ground-truth continuations. When the prompt comes from duplicate examples in the train set, XL-Original reproduces the groundtruth continuation over 40% of the time. XL-ExactSubstr and XL-NearDup still copy the groundtruth more often when the prompt comes from a duplicate example than when the prompt comes from a unique example, suggesting that more stringent deduplication may be necessary to remove memorization tendencies entirely.

Impact on Existing Models

Train-test leakage does not just impact models trained on C4. Table **??** shows that the presence of near-duplicates of the evaluation set in the train set has a significant impact on model perplexity for two standard models: Transformer-XL [**dai2019transformer**], which was trained on LM1B, and GROVER [**zellers2019defending**], which was trained on RealNews. For Transformer XL, the perplexity halves on examples identified as near-duplicates. For GROVER, the difference, though not quite as stark, is present in both model sizes considered.

Existing models also suffer from the problem of generating text from their train sets. We find that 1.38% of the tokens in the official release of 25k GROVER-Mega outputs are part of verbatim matches in RealNews of at least length 50. Likewise, more than 5% of the tokens in ~200k sequences outputted by GPT-Neo 1.3B [**gpt-neo**] are part of a 50 token matches of its training data, the Pile [12].

4.3.8 Discussion

The focus of this paper is on the datasets used to train language models. While recent work focused on documenting the potential harms that could arise from problematic datasets [**bender2018data**, **gebru2020datasheets**], less work has been done to quantitatively analyze properties of real language modelling datasets, like **dodge2021documenting** has done for C4. Our paper provides analysis on one particular axis, that of data duplication.

Our experiments measured what could be quantified: the amount of duplicate content in common datasets, the effect of deduplication on trained model perplexity, and the reduction of memorized content in trained models through deduplication. We do not focus on the nature of the data being removed by deduplication or memorized by LMs.

Privacy is an important subject for future work, as memorized training data has significant privacy consequences. By this, we mean the standard privacy definition that a model should not reveal anything particular to the specific dataset it was trained on, as opposed to another training dataset from a similar distribution [**shokri2017membership**].[17] Training on standard datasets that have not yet been deduplicated results in models that are particularly sensitive to examples that happened to be repeated multiple times, and this has negative privacy implications. For instance, it could violate a person's expectations of privacy if their publicly available personal data appeared in a different, surprising context. Downstream applications of LMs, such as the game AI Dungeon[18], should also not output memorized content like adverts for real products.

We stress that in our experiments, we do not distinguish between undesired memorized text (such as phone numbers), innocuous memorized text (common phrases), and text we may want to be memorized (such as a quote by a public figure), and instead treat all instances of the LM generating text that closely matches the training set as problematic. While we qualitatively observed that much of the identified memorized content was relatively innocuous, a more systematic study of the risks associated with the detected memorization was beyond the scope of this work.

We also do not investigate the negative consequences of deduplication. Some language tasks explicitly require memorization, like document retrieval or closed-book question answering. Also, text that gives attribution is often duplicated across documents, so removing duplicate substrings could correspond to removing *just* the attribution, which could result in models that learn the content without its attached attribution. Deduplication is also not sufficient to remove privacy-sensitive data like bank passwords and medical records which should never be used in training data.

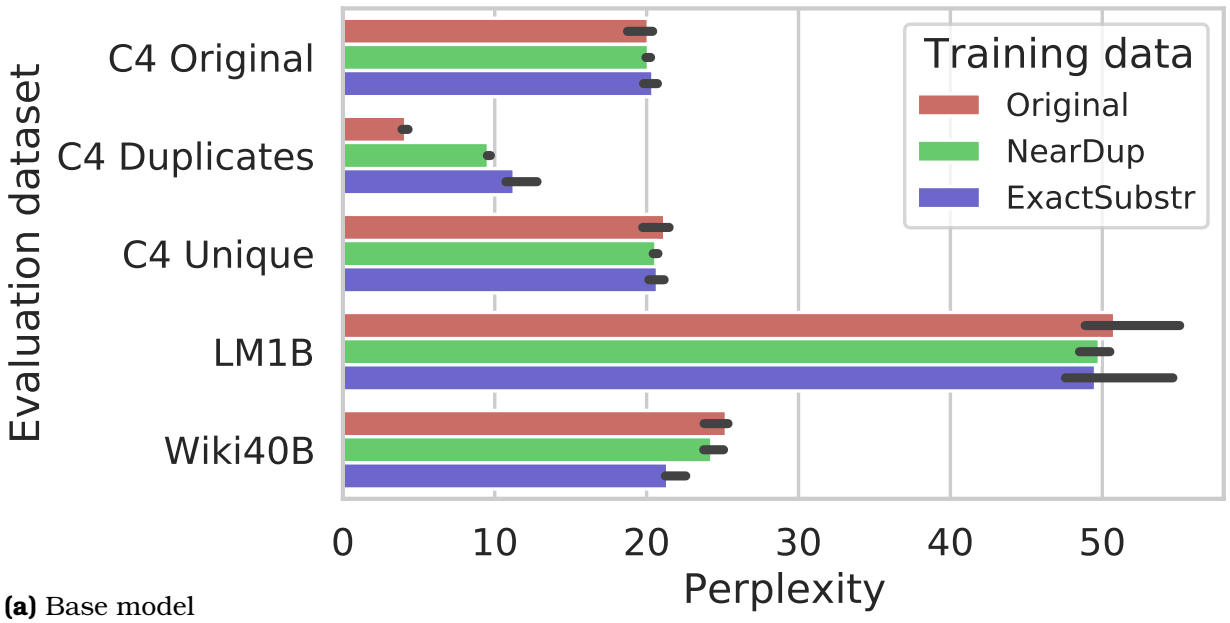
17 Another interpretation of privacy focuses on the sensitivity of the data involved, when a model is trained on and able to reproduce personal identifiers or other forms of "private data." Our definition is more expansive.

18 https://play.aidungeon.io/

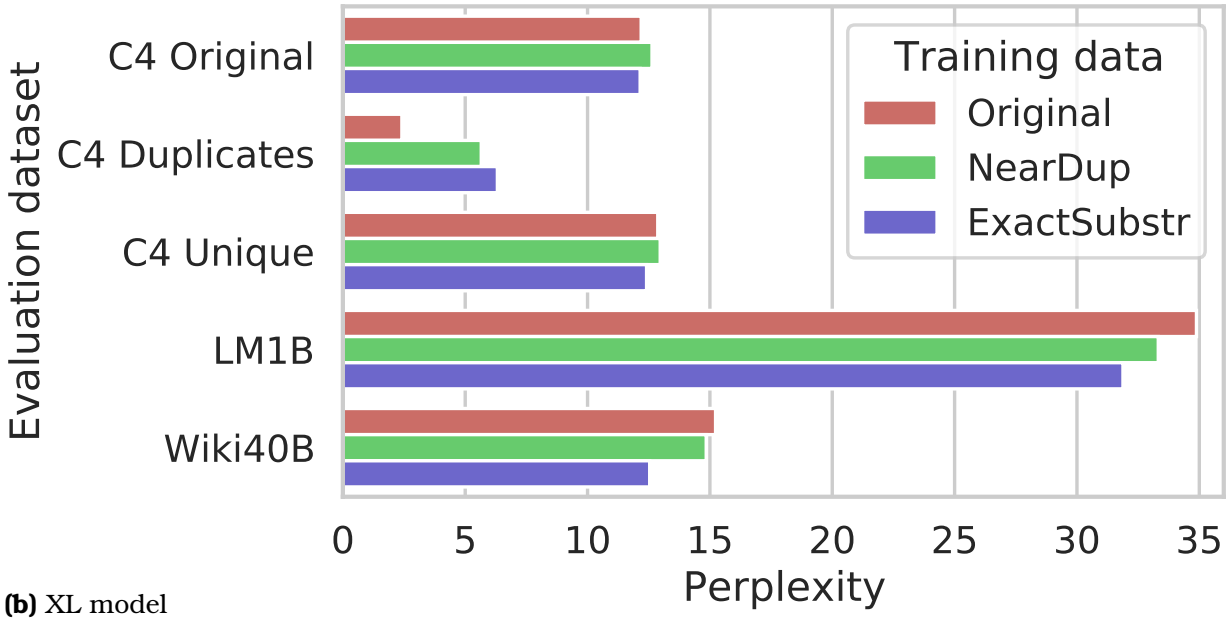
Ultimately, whether memorization is a desired property of a language model, or else risky and unwanted, depends on the nature of the text that has been memorized and on the downstream applications of the trained model. However, since the trend has been towards creating datasets and models that are application-agnostic, we encourage researchers to think carefully about the limitations of the data collected and the how the model's intended usage constrains what should be part of the training set. Developing techniques to memorize or forget specific sequences depending on the end application is a promising research direction.

We encourage future language model research to perform dataset deduplication, either by training on the deduplicated datasets we release, using the deduplication tools we release, or following our approach to deduplicate datasets with new tools.

The exact technique used to perform deduplication is less important than performing stringent deduplication in the first place. On the whole, deduplication does not harm, and sometimes improves, model perplexity, despite the fact that the deduplicated datasets are smaller and faster to train on. It is especially important that there are no duplicates between the training and testing sets, because overlap here explicitly encourages selecting models that memorize the training data. Lastly, deduplication helps to reduce some of the privacy concerns around LMs memorizing their training data.



(a) Base model



(b) XL model

Figure 4.2: Impact of deduplicating the training set on validation perplexity. In **(a)**, we plot the results from T5 base (110M parameters) across three training runs with different random initializations. The black bar represent the lowest perplexity to the highest perplexity, and the colored bar the median perplexity. In **(b)**, we plot the results from T5 XL (1.5B parameters). For C4, we evaluate on *C4 Original*, the original validation set; *C4 Unique*, a subset of the validation set identified by NearDup as having zero matches across C4; and *C4 Duplicates*, a subset of the validation set identified by NearDup as having a match in the C4 train set.

Text	Freq in C4
HD wallpaper. This wallpaper was upload at April 19, 2019 upload by admin in.You can download it in your computer by clicking resolution image in Download by size:. Don't forget to rate and comment if you interest with this wallpaper.	40,340
to the address posted below. Include our failure information form,a packing slip with your Company name, contact person, and Email address or phone number. Upon receipt of your repair, we\'ll inspect it and then contact you with a quote or evaluation notice. Normal turn aro und for repair is 5 to 7 business days, with "Rush Repair" available.	5,900
is a great place to begin your search. Whether you are a first-time home buyer or you are already familiar with the home buying process, you can be assured that you have the best tools and the perfect agent available to help with your	5,358
pics at these awesome group starting P letter. Desktop wallpapers were first introduced way back in the 1980s and have gained immense popularity since then. It is possible to come across more than 80 million sites on the web offering some sort of wallpaper.	848
flowers will let them know you're thinking of them and wishing them well. Cheerful yellow flowers bring their own sunshine and will get right to work on lifting spirits, and a colorful vase will bring loads of smiles to friends and visitors! Get Well flower arrangements from	479
our premier 24 hour emergency* plumbing and heating solutions. We realise that when your heating fails or pipes and drains leak it can cause havoc with your routine and even cause damage to your property. When a plumbing problem occurs that requires an immediate response we provide qualified local plumbers throughout	56
is to remove all images that violate copyrights. Please contact us to request that images be removed or to assign proper credit. The images displayed on this site may be used for Free or educational purposes only. If you would like to use any of the images displayed on this site for any other purpose, please obtain permission from the owner. www.	48
list of fishing locations, providing interactive maps that show each location's GPS coordinates, nearby facilities (like restaurants, gas stations, marinas and fishing shops), their current and forecasted weather and, if available, their water conditions.\nFind any of the 8	5
. Dyer, Ph.D., is an internationally renowned author and speaker in the field of self-development. He's the author of 30 books, has created many audio programs and videos, and has appeared on thousands of television and radio shows.	5

Table 4.6: A selection of substrings identified by ExactSubstr as being in C4 multiple times. The number of times this exact substring occurs in C4 is also given.

Table 4.7: A selection of substrings generated by XL-ORIGINAL with no prompting (and top-k with k=50) that were identified by EXACTSUBSTR as being in C4 multiple times. The number of times each substring was found in C4 is given. We observe that most memorized generations tend to be from advertisements.

Generated Text	Freq in C4
, you'll need to be knowledgeable to make the very best decisions. We will make sure you know what can be expected. We take the surprises from the picture by giving accurate and thorough information. You can start by talking about your task with our client service staff when you dial 888-353-1299. We'll address all of your questions and arrange the initial meeting. We work closely with you through the whole project, and our team can show up promptly and prepared.	5,497
then Waterside Lodge are well equipped for the task. Our fully equipped family sized lodges offer a comfortable luxurious stay for a fantastic price, giving you beautiful views of the lakes and the surrounding countryside. Offering luxurious self-catering holidays in our fully featured Scandinavian holiday lodges. Perfectly located to explore the beaches, coastline. All of our lodges are sized for 6 people and are furnished to the highest standards to ensure you have a stay like no other. At Waterside Lodge the stay itself is only half of the package, Waterside lodge is situated closely to the Heritage Coast which makes our lodges the perfect stay for anyone wanting to get away and have a relaxing countryside break from the city. Whilst you stay with us be sure to take advantage of all the activities Waterside Lodge has to offer. Such as the use of our on-site fishing lakes for the keen fisherman, free internet access, outside relaxation areas, comfortable lounges and much more.	571
you are only looking to find rent to own homes in your city or are open to exploring all kinds of rent to own home listings, our database does it all. One of the best aspects of iRentToOwn.com is that, besides options to rent to buy a house, it has numerous other categories of home sale options. These include bank foreclosure homes, pre-foreclosure homes, short sales, HUD/government foreclosures, auction homes and owner-financing/FSBO (For Sale By Owner) homes. With help from the convenient search features offered by our site, shoppers are able to find their ideal lease to own home, real estate company, and more in South	51
, IL employs journeyman as licensed to work by themselves, without direct supervision, installing wiring, outlets and fixtures. Our journeyman also does service work, troubleshooting when a breaker fails or a light stops working. Our journeyman does not offer permits that must be issued by our master. Our journeyman follows our master's plans and directions. Our journeyman's responsibilities will vary based on the work that needs to be done. Our journeymen are skilled with residential, commercial and industrial installations and repairs.ust work from six years as an apprentice, under direct supervision of our master, and pass a journeyman test. This person also must have some classroom education on the National Electrical Code and fundamental electricity in a technical school a program affiliated with the National Joint Apprenticeship Training Council. Journeyman training combines hands-on work with education on basic electricity.	6
combustion process of a petrol engine is never perfect. Dangerous gases, such as nitrogen oxide, carbon monoxide and hydrocarbons will arise and it is the job of the catalytic converter to reduce these to safer emissions. These cat converters can fail by becoming clogged, or if the engine has bad exhaust valves or the plugs fail, causing unburned fuel to overheat the converter. Mettam's Mufflers can resolve these issues with your Karr	5

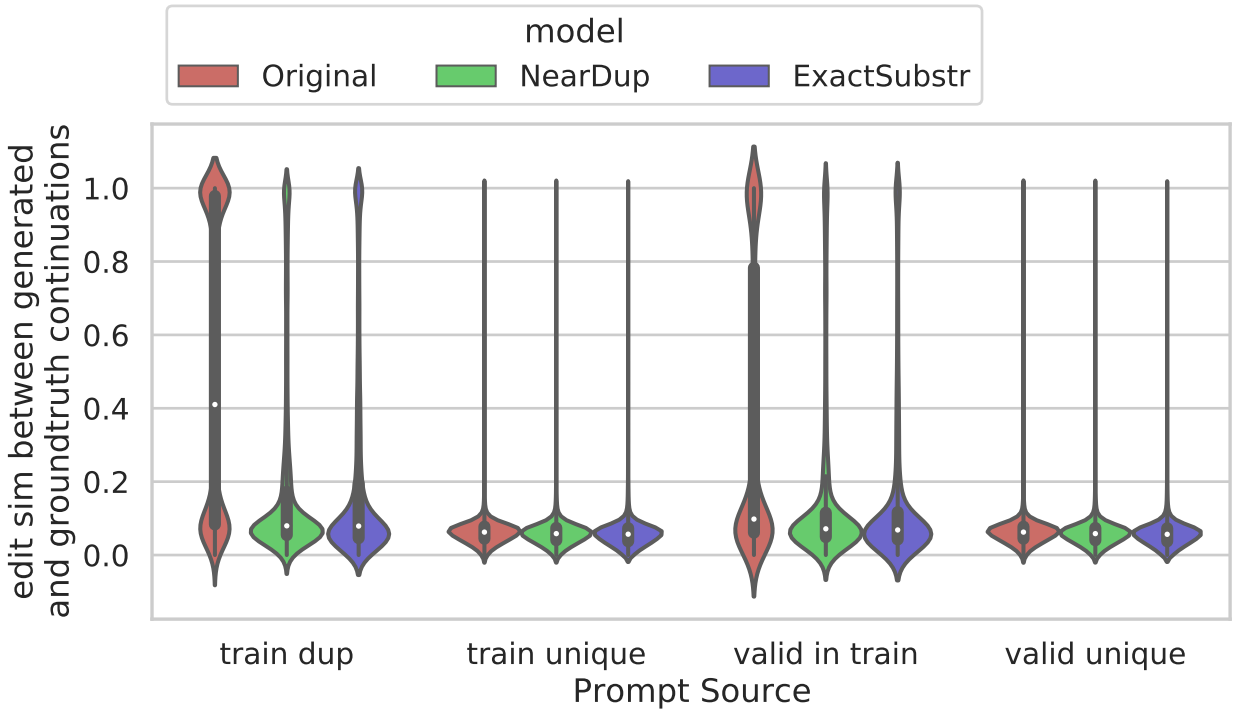


Figure 4.3: Memorized continuations distribution

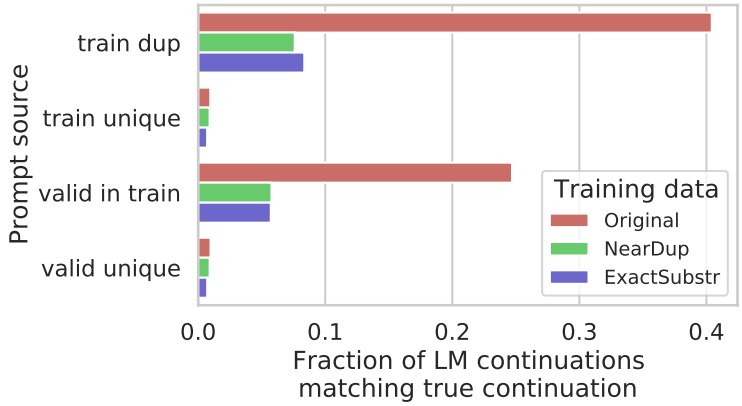


Figure 4.4: The proportion of generations which have edit similarity above 0.8 with the groundtruth continuation when using the LM to generate continuations for 32-token prompts identified by NEARDUP as either duplicated or unique.

Table 4.8: For each model, the perplexity of the official validation set (*Orig*), valid set examples which were identified by NEARDUP as matches of train set examples (*Dups*), and valid set examples identified by NEARDUP as unique (*Unique*). Due to the size of the RealNews validation set, we evaluated on only the first 25k examples meeting each condition.

Model	Dataset	Orig	Dups	Unique
Transformer-XL	LM1B	21.77	10.11	23.58
GROVER-Base	RealNews	15.44	13.77	15.73
GROVER-XL	RealNews	9.15	7.68	9.45

5 ENABLING APPLICATIONS IN CREATIVE WRITING

One application where NLG has considerable potential is in the development of tools for creative writing. AI-assisted creative writing is an attractive testbed NLG systems because ideation tools are already part of writers' arsenal, and mistakes like hallucinating false facts are less problematic in fiction than in domains like automatic news summarization, where faithfulness to the real world is crucial. In addition, writers have been grappling with the concept of sentient, human-like machines for at least as long as computer scientists have.

In this chapter, I describe work I have done toward bridging the gap between what most language models do by default (predict a continuation for a prompt) and the operations writers actually would want. First, I will describe efforts to capture longer-term coherence by building a language model that operates over sentences rather than other sub-words. Second, I will show how existing neural networks can be modified to support fill-in-the-blank style tasks in addition to the more common paradigm of continuation. Filling in the blank is a common control that is requested by writers. Third, I will present a recipe for performing sentence style transfer into arbitrary styles–such as rewriting text to be more Shakespearean, metaphorical, or meladramatic–without any exemplars of the task or task-specific model training.

To test out how these and other NLG-based tools can be used in practice, we built Wordcraft, a word processor augmented with a variety of "smart" writing controls and

suggestion tools. I will end by describing the features of Wordcraft and the results of a user study conducted with the tool.

5.1 MOTIVATION

Artificial intelligence systems which can write stories have been a goal of researchers since the early days of computing. However, much of the early work in this area was focused on building systems which could produce an entire story from scratch given an initial set of constraints.

Rather than focusing on developing NLG systems which produce entire stories, my research in this area focuses on enabling better human-AI collaboration.

5.2 TOWARDS SENTENCE-LEVEL LANGUAGE MODELS

5.3 MODELS FOR INFILLING TEXT

5.3.1 Motivation

Natural language generation systems are increasingly being incorporated into applications where a human writer and an AI jointly collaborate to construct text. Wordcraft, The AI-assited text processor I describe in Section **??** is one such application. Another is Storium, where players of a writing game, have the option to accept suggestions from a natural langueg generatin system [**akoury2020storium**]. There are also more practical domain such as email composition assistance and code synthesis [**buschek2021impact**,

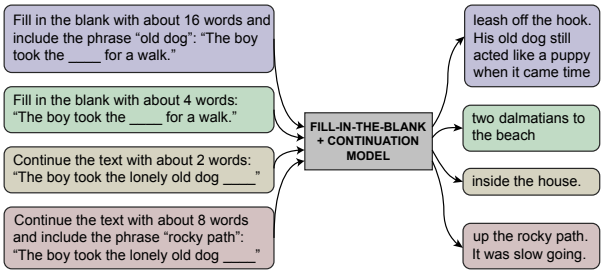


Figure 5.1: A single model that can handle a variety of related writing tasks is more efficient than separate models per task.

wu2018smart, **austin2021program**]. Many of these applications are limited to generating text at the end of what has been written so far. This is because both historical language models (LMs) and state-of-the-art neural LMs are typically designed to produce text by repeatedly predicting the next word in a sequence given the previous words. However, there is a need for more powerful interactive tools which enable writers to solicit insertions at any chosen position within the existing text, a task variously referred to as fill in the blank (FrrB), infilling, or the Cloze task [**taylor1953cloze**]. For example, a creative writer might want a tool which can insert a description of a place or character, and a programmer might want a system that can fill in a method in the middle of their code.

Most prior work tackling FrrB consider it a separate task from continuation, one to be specifically optimized for, for example training a custom model from scratch [**ippolito2019unsupervised**, **zhu2019text**, **mori2020finding**], finetuning a model trained originally for continuation [**donahue2020enabling**], or using a combination of pre-trained models [**huang2020inset**]. Having separate trained models for FrrB and for continuation is inefficient for downstream applications where maintaining multiple neural networks can b e prohibitive.

Any model that can do FrrB can be made to do continuation simply by placing the blank at the end of the input. Thus, in this section, we describe how models trained on FrrB can be employed effectively for both infilling and continuation operations. We show how T5 [39], one of the most popular pre-trained models, can reasonably handle both tasks, as it

Table 5.1: Examples of the finetuning objectives. “8" is the approximate length in words of the target sequence. During finetuning, about 25% of training examples took each of these formats.

Example Type	Input	Target
C4FillBlank no goal	fill: I love avocados. I ate a sandwich covered in them. _8_ I talked to my doctor about it later. It turned out I was allergic to avocados.	After I ate it, my mouth was itchy and tingly.
C4FillBlank with goal	fill: I love avocados. I ate a sandwich covered in them. _8_ I talked to my doctor about it later. It turned out I was allergic to avocados. **G**oal: mouth was itchy	After I ate it, my mouth was itchy and tingly.
C4FillBlank no goal	fill: I love avocados. I ate a sandwich covered in them. After I ate it, my mouth was itchy and tingly. I talked to my doctor about it later. _8_	It turned out I was allergic to avocados.
C4FillEnd with goal	fill: I love avocados. I ate a sandwich covered in them. After I ate it, my mouth was itchy and tingly. I talked to my doctor about it later. _8_ **G**oal: allergic to	It turned out I was allergic to avocados.

was pre-trained with a FitB-like objective. Finetuning T5 further improves its ability and also allows for the incorporation of controllability of generation length and word choice.

5.3.2 Supporting FitB and Continuation

We define filling in the blank as the task of predicting text to replace a single missing span, usually demarcated with a special token, in an input text passage. (Some prior work considers inputs with multiple blanks, but inserting text at one position at a time better matches the kinds of edits humans do.) We define continuation in the traditional language modeling sense–predicting the next token in a sequence given only the previous tokens. **donahue2020enabling** discuss how language modeling is a special case of infilling, and they use this as justification to finetune a continuation-based language model to do infilling. However, we argue that if continuation is a subtask of infilling, it makes more

sense to go in the opposite direction: prioritize a model which can do infilling and check that it achieves satisfactory performance at continuation.

T5 is a model pre-trained with a "span corruption" objective very similar to FrrB; the model is asked to reconstruct the missing text after random sub-sequences of the input are replaced with special identifiers. Thus, a pre-trained T5 model can be used without any further training to do both continuation and infilling by appropriately choosing text to mask out. The encoder-decoder architecture of T5 is also more conducive to FrrB than decoder-only architectures like GPT-2 [38] which are typically used for continuation-based language models. This is because the attention mechanism in encoder-decoder architectures allows the context on the left side of the blank to attend to the context on the right, while decoder-only architectures only support masked attention (each token can only attend to the positions to its left).

Even though T5's pre-training objective was a form of FrrB, finetuning is still advantageous. For one, our definition of FrrB only includes a single masked out substring, not multiple, so finetuning improves alignment with the goal task. Finetuning also allows us to incorporate additional conditioning signals not supported by the pre-trained T5, such as being able to specify the desired length of the generated text or specify words that ought to be included in the blank, a task we refer to as "goal conditioning." Length control, which comes by default in a traditional language model by simply sampling more or fewer tokens, is particularly necessary for FrrB, where the end of the generation must fit seamlessly with the text to its right.

The biggest language models available today were largely trained in the continuation rather than the FrrB paradigm [**gpt3**, **gpt-neo**]. Since our primary goal is to have a single model for both tasks, we also address the question: if a continuation-trained model is big enough, can it handle FrrB without the need for finetuning? Few-shot learning with large language models, as popularized by **gpt3**, has had success on many tasks in NLP.

We try out this approach for FrtB by designing a few-shot prompt containing several demonstrations of the FrtB task, formulated in a similar “infilling by language modelling" style as **donahue2020enabling**.

5.3.3 Experimental Setup

For all primary experiments, we use the 800M parameter v1.1 ‘large’ model. We also show soome addditional results comparing against the 3B parameter ‘XL’ T5 model. To finetune T5 for FrtB, we construct training examples from documents by first partitioning the document text into a left context, gap, and right context. The input sequence is then the left and right contexts concatenated with textual representations of the additional conditioning signals. The target sequence is the true text for the blank. This formulation easily supports continuation, as the blank can be deliberately placed at the end (i.e., providing no right context). Documents are drawn from C4, the same dataset T5 was pre-trained on. Documents are split into word sequences, and these are then randomly truncated to be between 256-512 words long. A substring of between 1 and 64 words is selected to be blanked out. For half of the training examples the blank is randomly selected, and for the other half it is always placed at the end. To support length conditioning, we follow **roberts2020exploring** and include a bucketed version of the target length as part of the blank. To support goal conditioning, for half the examples, a random substring of up to half the words of the target is appended to the end of the input. Examples are shown in Table **??**.

We compare T5 against a state-of-the-art 137B parameter decoder-only language model (Llm) trained explicitly for continuation and used successfully for few-shot learning in other domains [**austin2021program**, **reif2021recipe**]. This model is used (1) as a standard

Table 5.2: Perplexity of evaluation sets according to LLM when the blank has been filled with approaches involving no fine-tuning (top), finetuned approaches (middle), and the groundtruth (bottom). Lower values indicate that the text was considered more fluent by the LLM.

	C4FILL BLANK	RWPFILL MIDDLE	ROCFILL BLANK
Few-shot LLM	14.14	19.48	18.21
Pre-trained T5	10.38	14.08	22.62
Finetuned T5	10.33	14.08	20.47
donahue2020enabling	N/A	N/A	23.28
Groundtruth	9.41	12.99	16.90

continuation model, prompting with only the left context of an example; and (2) in a few-shot learning paradigm.

We evaluate continuation and FITB on C4 as well as two story writing datasets, as creative writing assistant applications are one of the key areas we expect to benefit from multi-task models [**wordcraft**]. Reddit Writing Prompts (RWP) is a corpus of stories from the 'r/WritingPrompts' sub-Reddit [**fan2018hierarchical**], and we construct validation sets RWPFILLBLANK and RWPFILLEND using the same method described in the previous section. C4 and RWP validation sets are capped to 5,000 examples. ROC Stories (ROC) is a crowd-sourced dataset of five-sentence commonsense stories [**mostafazadeh2016corpus**]. For ROC Stories, the 2018 validation set is used to construct ROCFILLMIDDLE, where the middle sentence of each story is blanked out, and ROCFILLEND, where the last sentence is blanked out. Unless otherwise noted, all evaluation is done without goal conditioning and uses random sampling with top-k=50 as the decoding strategy. Example generations for all evaluation sets can be found at https://bit.ly/2U0Ixxa.

Table 5.3: Perplexity of continuation-based evaluation sets when a continuation has been generated using approaches with no finetuning (top) and two settings of finetuning T5 (middle).

	C4Fill End	RwpFill End	RocFill End
Pre-trained T5	10.09	13.51	21.71
T5 FillBlankCont	10.04	13.74	19.60
T5 Lm-Adaption	10.06	13.71	19.68
Groundtruth	9.41	12.99	16.90

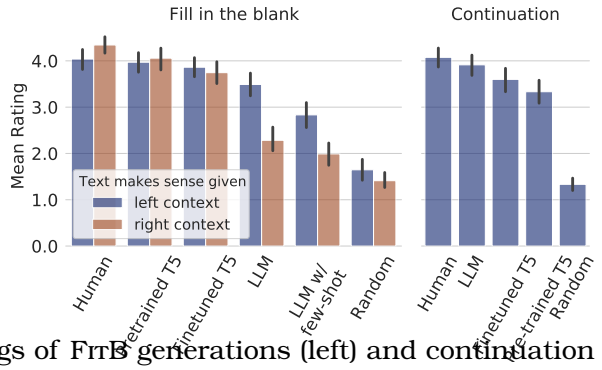


Figure 5.2: Human ratings of FitB generations (left) and continuation generations (right). Error bars are 95% confidence intervals.

5.3.4 Results

Difficulty in Developing a Few-Shot Prompt for the Infilling Task

Filling in a blank seems like a task that ought to be easy to accomplish with few-shot learning techniques. Training data for large language models often contains fill-in-the-blank style examples, as school lessons with cloze-style questions are relatively common on the internet. Furthermore, infilling ought to be an easier task tha continuation since there is more information available for the model to base its prediction on. However, after conducting a large-scale study of many possible few-shot prompts, we found that this technique fell short for the fill-in-the-blank task.

Table 5.4: Accuracy of models finetuned on FillBlankCont at correctly using provided length and goal conditioning signals.

Finetuned T5	Context	Length
C4FillBlank	0.860	0.877
RwpFillBlank	0.797	0.881
C4FillEnd	0.858	0.775
RwpFillEnd	0.791	0.746

Choosing appropriate examples for a few-shot prompt is very challenging because task performance is often sensitive to minor changes in prompt design [**zhao2021calibrate**]. We experimented with prompts randomly selected from the C4, Reddit Writing Prompts, and ROC Stories training sets, as well as prompts consisting of examples we hand-wrote with the goal of story-writing in mind. For each prompt source, we randomly generated five possible prompts, each with three examples, using the method descrribed in Section **??**. To simplify the task, we conditioned on desired length but did not include goal conditioning.

An example prompt is shown in Figure **??**. When choosing random few-shot prompts from the dataset train sets, in order to keep the few-shot prompt text within the 512-token context length limit of model we used for inference, we only considered examples that contained 100 or fewer tokens, so that the max length of the few-shot prompt was no more than 300 tokens. This left 212 tokens for the text of the actual example we were interested in performing the FitB task on. For each evaluation set, examples with inputs longer than 212 tokens were excluded from analysis. For our hand-written prompt, we wrote the 7 examples shown in Table **??**. We generated 5 possible prompts by randomly subsampling 3 examples out of these 7.

Table **??** shows the perplexity of the generations from each few-shot prompt. We note that even leaving room for 212 tokens worth of context text, some evaluation examples did not fit in the prompt length, and these examples were skipped when doing this analysis.

Figure **??** shows a histogram of the fraction of validation set examples that remained for each few-shot prompt after the too-long examples were filtered out. Based on these results, we chose to include in human evaluation the best few-shot prompt from from RocFillMiddleand the best few-shot prompt from C4FillBlank. Figure **??** in the main paper shows the result from the C4FillBlankfew-shot prompt, whose outputs were rated slightly higher by human annotators.

Unfortunately, it is computationally impossible to conduct an exhaustive search of all possible example choices and prompt formats. While none of the prompts we tried led to strong performance at the fill-in-the-blank task, we cannot rule out the possibility there might exist a prompt we did not test for which this performance might have been significantly better. For example, we did not conduct formal experiments to systematically vary the prompt wording/formatting shown in Figure **??**. We can conclude that the process of finding an ideal prompt requires time-consuming trial-and-error and is quite difficult!

Table 5.5: Perplexity of evaluation sets when the blank has been filled in using Llm with few-shot prompting (top) and our best fine-tuned T5 model ((bottom). Among the few-shot results, the best method for each dataset is bolded, as well as methods within one standard error.

Few-shot source:	C4Fill Blank	RocFill Middle	RwpFill Blank	RwpFill Blank-Sent
C4FillBlank	15.67	19.72	**19.65**	**16.82**
RocFillMiddle	**14.14**	19.61	**19.48**	**16.36**
RwpFillBlank	24.39	20.29	32.33	28.13
RwpFillBlank-Sent	18.91	**18.21**	24.44	19.87
FS Custom	17.98	19.80	21.72	18.38
Finetuned T5 XL	9.99	19.00	13.64	10.03
Finetuned T5 Large	10.33	20.47	14.08	10.37

Automatic Evaluation

We measure the fluency of proposed generations by evaluating the perplexity of each dataset's examples when the predicted text is placed in the blank [**donahue2020enabling**]. We use the LLM to measure perplexity[19]. The results are shown in Table **??**. We see that the LLM struggles to generate fluent infills, even when used in a few-shot setting. The only exception to this is ROC Stories, a dataset with fairly simplistic, predictable language. Finetuning T5 does not result in significantly improved fluency over the pre-trained model except on ROC Stories. Lastly, for ROC Stories, we compare against **donahue2020enabling**'s finetuned GPT-2 small, which yielded less fluent predictions. Table **??** shows a similar analysis on our continuation-style datasets. Both T5-based models achieve roughly the same fluency.

Human Evaluation

Human evaluation was conducted on 70 examples, 35 from RWPFILLBLANK and 35 from RWPFILLEND, with examples about evenly distributed across length buckets. For RWPFILLBLANK evaluation tasks, the rater was presented an input context and several possible sequences that could go in the blank. They were asked to rate each sequence first, on how well it fit the text before it, and second, on how well it fit with the text following it, according to a 5-point slider For RWPFILLBLANK, the task was almost the same, except that the rater was presented only a left context and asked to rate how well it continued the prompt. A screenshot of the Human Intelligence Task (HIT) used for annotations is shown in Figure **??**. Workers were paid originally paid $1.85 per HIT, but since the average HIT duration ended up being 15 minutes, we awarded each rater a bonus to raise their pay to an average of $10 per hour. Each example was shown to three raters, and

19 Note, since this is the same model being used for generation for our continuation baseline, this metric may be biased.