for the model is not and no details have been released regarding the makeup of this data.

MPT 30B (MosaicML, 2023) is a decoder-only model trained by Mosaic and is the first model released by the company. Released on June 22nd 2023, MPT 30B has an 8k context window and outperforms GPT-3 on various reasoning tasks. Training data consists of deduplicated C4 (Raffel et al., 2020; Lee et al., 2022), the RedPajama[10] split of CommonCrawl, and selected programming languages from The Stack (Kocetkov et al., 2022).

Cohere *command* (Cohere, 2024) is a closed-source model trained and released by Cohere. While original versions of this model were quoted as having roughly 50 billion parameters (Liang et al., 2023b), the size and training data of the current version of the Cohere model is unknown. Unlike OpenAI, Cohere does not version the *command* model and so, much like with `text-davinci-002` we are unable to expand our dataset to new domains without re-generating all generations. We queried the command model through both the `co.generate()` and `co.chat()` endpoints from November 1st to November 2nd 2023.

E.3 Prompts

In Table 12 we report the prompt templates for each of the 8 domains used in our project. We specifically avoided biasing the model towards a particular length or style of generation and did not use any of the text from the human-written documents other than the title when prompting.

To decide on the particular form of our prompts, we conducted two rounds of manual review. These rounds consisted of the authors determining problematic prompts by manually reviewing 10 generations per model in each domain for instances of degenerate repetition, meta-commentary or other signs of generated output. In problematic domains, we conducted individual explorations to determine if there existed some high level prompting concept that removed the unintended behavior and whether or not using such a prompt style caused regressions across other generators. After identifying a desired prompting change, we re-wrote our templates and restarted the review process.

Through this investigation, we found that continuation-style models benefit greatly from explicitly stating the source website in the prompt and that chat-style models benefit from language explicitly asking them not to repeat the title of the article (See Table 12). While we were unable to remove all instances of degenerate repetition or meta-commentary, this investigation significantly increased the quality of our dataset and we encourage future work on dataset creation to conduct a similar process when engineering their prompts.

[10] `https://www.together.ai/blog/redpajama-data-v2`

Attack	θ	Source
Alternative Spelling	100%	(Liang et al., 2023c)
Article Deletion	50%	(Liang et al., 2023a; Guerrero et al., 2022)
Homoglyph	100%	(Wolff, 2020; Gagiano et al., 2021)
Insert Paragraphs	50%	(Bhat and Parthasarathy, 2020)
Number Swap	50%	(Bhat and Parthasarathy, 2020)
Paraphrase	100%	(Krishna et al., 2023; Sadasivan et al., 2023)
Misspelling	20%	(Liang et al., 2023a; Gagiano et al., 2021; Gao et al., 2018)
Synonym	50%	(Pu et al., 2023a)
Upper Lower	5%	(Gagiano et al., 2021)
Whitespace	20%	(Cai and Cui, 2023; Gagiano et al., 2021)
Zero-Width Space	100%	(Guerrero et al., 2022)

Table 11: The adversarial attacks used in the project. θ represents the manually determined fraction of available attacks carried out. We determine this fraction through manual review.

E.4 Adversarial Attacks

In Table 11 we list the attacks used along with the attack rate θ and the relevant sources for the attacks. In this section we will list the attacks in more detail and discuss the various design decisions made. Implementations for all attacks can be found in our GitHub repository.

Alternative Spelling We use an American to British English dictionary[11] to construct a mapping between American and British spellings of words. We then find all instances of such words in the generation (defaulting to the longest available match if there were multiple substring matches for the same token). We then randomly sample a fixed percentage θ of the possible mutations to make with a set seed and apply the attack at those indices.

Article Deletion We search through the text and find every instance of the articles "a", "an", and

[11] `https://github.com/hyperreality/American-British-English-Translator`

	Continuation-Style Prompt	Chat-Style Prompt
Abstracts	The following is the full text of the abstract for a research paper titled "{title}" from arxiv.org:	Write the abstract for the academic paper titled "{title}".
Books	The following is the full text of a plot summary for a novel titled "{title}" from wikipedia.org:	Write the body of a plot summary for a novel titled "{title}". Do not give it a title.
Code	The following is the full text of a Python code solution to the exercise "{title}" from stackoverflow.com:	Write just the Python code solution for the problem "{title}".
German	Schreiben Sie einen Nachrichtenartikel mit dem Titel "{title}".	Es folgt ein Nachrichtenartikel mit dem Titel "{title}":
Czech	Následuje článek s názvem "{title}":	Napiš text článku, který má nadpis "{title}".
News	The following is the full text of a news article titled "{title}" from bbc.com:	Write the body of a BBC news article titled "{title}". Do not repeat the title.
Poetry	The following is the full text of a poem titled "{title}" from poemhunter.com:	Write the body of a poem titled "{title}". Do not repeat the title.
Recipes	The following is the full text of a recipe for a dish called "{title}" from allrecipes.com:	Write a recipe for "{title}".
Reddit	The following is the full text of a post titled "{title}" from reddit.com:	Write just the body of a Reddit post titled "{title}". Do not repeat the title.
Reviews	The following is the full text of a review for the movie "{title}" from IMDb.com:	Write the body of an IMDb review for the movie "{title}". Do not give it a title.
Wiki	The following is the full text of an article titled "{title}" from wikipedia.com:	Write the body of a Wikipedia article titled "{title}".

Table 12: The text of the generation prompts for all ten datasets in both continuation and chat style. The field {title} was replaced with the title of the book, dish, or news article before being passed into the generative model.

"the". We then randomly sample a fixed percentage θ of the possible mutations to make with a set seed and apply the attack at those indices.

Homoglyph Homoglyphs are character that are non-standard unicode characters that strongly resemble standard English letters. These are typically characters used in Cyrillic scripts. We use the set of homoglyphs from Wolff (2020) which includes substitutions for the following standard ASCII characters: a, A, B, e, E, c, p, K, O, P, M, H, T, X, C, y, o, x, I, i, N, and Z. We limit ourselves to only homoglyphs that are undetectable to the untrained human eye, thus we are able to use an attack rate of $\theta = 100\%$ and apply the attack on every possible character. For characters that have multiple possible homoglyphs we randomly choose between the homoglyphs.

Insert Paragraphs For this attack, we again split sentences using Punkt (Kiss and Strunk, 2006) and construct a list of all inter-sentence spans. We then sample θ percent of those spans and add the double newline character `\n\n` in-between the sentences to simulate a paragraph break.

Number Swap For this attack we use the following regular expression to extract all instances of numerical digits in the generation: `"\d+.?\d*"`. We then randomly select θ percent of these digits to modify and for each digit we randomly select an alternate number between 0 and 9 to replace the character with.

Paraphrase For paraphrasing we run the DIPPER-11B model[12] from Krishna et al. (2023) through HuggingFace (Wolf et al., 2020). DIPPER is a fine-tuned version of T5-11B (Raffel et al., 2020) specifically made for paraphrasing text to avoid machine-generated text detectors. We use the default settings from the paper, namely a sentence interval of 3 with lexical diversity of 60 and order diversity of 0. Since paraphrases are not inherently noticeable when models are correct, we are able to apply this attack across the entirety of the output text ($\theta = 100\%$).

Misspelling For this attack, we manually constructed a dictionary of common misspellings[13] and only applied the attack to instances of words that have misspellings in our dictionary. We did this to minimize suspicion from human readers, as

[12] https://huggingface.co/kalpeshk2011/dipper-paraphraser-xxl

[13] https://en.wikipedia.org/wiki/Commonly_misspelled_English_words

certain common words such as 'the' or 'him' are rarely misspelled. Instead of randomly selecting θ percent of the possible candidate words to misspell, we follow Gagiano et al. (2021) and misspell only the top θ percent most likely candidate words by log likelihood as determined by GPT 2 small. This allows us to choose only the most effective misspellings to apply.

Synonym For this attack we originally planned to use the DFTFooler algorithm from Pu et al. (2023a). However, we found the candidate synonyms to be of low quality and easily detectable by our manual analysis. Thus we decided to implement our own algorithm based largely on DFTFooler. Our algorithm produces high-quality and diverse synonym substitutions without relying on any of the large decoder-only language models used for generation.

We start by iterating over all tokens in the generation. For each token i we replace it with a mask token and get the top 20 most likely mask-fill candidates[14] according to BERT (Devlin et al., 2019). We then compute the part-of-speech tag for each candidate using NLTK (Bird and Loper, 2004) and reject all candidates that do not match the part-of-speech of the original token. We then get the static FastText (Bojanowski et al., 2017) embeddings for all candidate substitutions and reject all tokens that have cosine similarity of less than 0.5 with the original token. Doing this for each index i gives us a global list of all valid candidate swaps across the entire generated passage. From this list we select the $\theta \cdot L$ most likely synonym swaps according to BERT where L is the length (in tokens) of the passage.

The full code for this algorithm can be found in our project repository along with the implementations of the other adversarial attacks.

Upper-Lower This attack randomly selects some θ percent of the tokens in the passage and swaps the first letter of the token to be uppercase if it was lowercase and lowercase if it was originally uppercase.

Whitespace This attack randomly selects some θ percent of inter-token spaces and adds an extra space character inbetween the tokens. This can occasionally result in multiple spaces added between two tokens as sampling is done with replacement.

[14] In order to reduce computation time we limit BERT to a window of 20 tokens on either side of the index when determining the mask-fill candidates. We found no reduction in candidate quality from this modification.

Zero-Width Space The unicode zero-width space `U+200B` is a character that exists in text encoding but is not visible to human readers in most scenarios. Thus, for this attack we insert this character at every possible opportunity (before and after each visible character in the generation).

E.5 Repetition Penalty vs. Frequency Penalty vs. Presence Penalty

Given a temperature $T > 0$ and a set of scores $x_i \in \mathbb{R}^d$ for each token i in a vocabulary, the probability p_i of predicting the ith token is given by:

$$p_i = \frac{\exp(x_i/T)}{\sum_j \exp(x_j/T)}$$

The repetition penalty defined by Keskar et al. (2019) modifies this distribution as follows:

$$p_i = \frac{\exp(x_i/(T \cdot I(i \in g)))}{\sum_j \exp(x_j/(T \cdot I(j \in g)))}$$

Where g is a list of previously generated tokens and $I(c) = \theta$ if c is True else 1. OpenAI implements a form[15] of this penalty (referred to as a 'presence penalty') which is additive instead of multiplicative:

$$p_i = \frac{\exp((x_i/T) - I(i \in g))}{\sum_j \exp((x_j/T) - I(j \in g))}$$

Cohere (as of May 13th 2024) provides no documentation on the nature of their presence penalty despite requests from the authors for the proper documentation.

E.6 Hardware

We ran our generations over the course of 15 days from November 1st 2023 to November 15th 2023 on 32 NVIDIA 48GB A6000 GPUs. We ran all models with 16-bit precision as we found that outputs were identical to full precision and it cut our inference time in half. The amount of GPU hours used by each family of models is as follows: LLaMA 2 70B (+ Chat) 8,376 hours, MPT (+ Chat) 5,440 hours, Mistral (+ Chat) 672 hours, GPT2 (+ Chat) 352 hours. In total we used 14,872 GPU hours (620 GPU days) to generate the RAID dataset.

[15] https://platform.openai.com/docs/guides/text-generation/parameter-details