Training Data Extraction Attacks on Large Language Models: A Deeper Look into GPT-2 XL, GPT-2 IMDB, and LLaMA

Arti Schmidt
arthurts@princeton.edu

Brian Lou
blou@princeton.edu

Jenny Sun
jwsun@princeton.edu

Abstract

Training data extraction attacks are a way for adversaries to obtain training data from a language model by querying it. Such attacks are of increasing concern as language models become more powerful and widespread, because they can cause the leakage of personally identifiable information or private data. This paper reproduces an attack proposed by Carlini et al., who evaluated their approach on GPT-2. They found that larger models tended to be more susceptible to the attack, so we also evaluate a smaller model, GPT-2 IMDB, and a larger model, the state of the art LLaMA, to test this claim. Additionally, we introduce a new scoring metric for performing the attack based on DetectGPT. We find that larger models do tend to memorize more training data, and that the memorized data is qualitatively similar to that of the original data. We find that DetectGPT does not perform as well as the original scoring metrics, and discuss some limitations of the study that may have contributed to this. Code is available in our GitHub Repository.

1 Introduction

A language model is a probability distribution over sequences of words. They have a wide variety of applications, including text generation. In recent years, neural networks of unprecedented size with many billions of parameters and with new architectures like Transformers have achieved huge success. Models like OpenAI's GPT-4 and ChatGPT are now very well known and heavily used (Brown et al., 2020), and many companies are racing to catch up and build their own large language models (Touvron et al., 2023; Chowdhery et al., 2022; Hoffmann et al., 2022). While these models could have a great positive impact, it is important to investigate their shortcomings to protect user safety and privacy.

One such concern is known as a training data extraction attack (Carlini et al., 2021). In this attack, an adversary retrieves text from the model that is present verbatim in its training data. This is known as "memorized" text. In this paper, we are concerned with the scenario where the adversary has black box query access to the model, the same as considered by (Carlini et al., 2021). That is, the attacker can only obtain the probabilities of next words following some prompt. This is the same level of access provided by many popular large language model APIs today, which means that the attack is practical and widely applicable.

Large language models (LLMs) require a large amount of training data, which is often taken from public sources, like data scraped from pages on the internet. Any of this data is theoretically accessible through a training data extraction attack. Even if the data is public, there are still privacy concerns associated with its retrieval. For example, personally identifiable information that is present on just a handful of obscure public pages may not be much of an issue, but could become a problem if memorized and leaked by a language model to a wide audience. Furthermore, the problem could be much worse for language models that are trained partly or entirely on private data. They would have the potential to leak information that would otherwise be inaccessible.

In this paper, we reproduce results from (Carlini et al., 2021), in which a training data extraction attack with several variations is applied to the GPT-2 XL model (Radford et al., 2019). In addition, we apply the attack to the GPT-2 IMDB model, which is a version of GPT-2 that is fine tuned on a movie review dataset (gpt), and the LLaMA 7B model, which is a general 7 billion parameter LLM achieving performance on par with GPT-3 (**?**). We choose these models to explore the effectiveness of the method on a model fine-tuned to a relatively small dataset, and on a state of the art model that is larger than any used in the original paper. We also develop and test a modification of their attack that makes use of DetectGPT (Mitchell et al., 2023), an approach for determining whether given text came from a particular language model, in hopes of achieving comparable performance to the original methods. We open source our code in a GitHub repository [1].

We find that larger language models contain more memorized content and are thus more susceptible to data extraction attacks. Additionally, we find that our proposed scoring metric using DetectGPT did not perform as well as our baseline perplexity scoring metric, and attribute this a few reasons including the unnatural generation of fewer natural, human written language and more machine-like text such as code.

2 Related Work

Our paper was inspired by the work *Extracting Training Data from Large Language Models* by Nicholas Carlini, Florian Tramer, Eric Wallace from Google, Stanford, UC Berkeley, and more (Carlini et al., 2021). The original paper demonstrates how sequences of training data can be extracted from GPT-2 by sampling text and then predicting the likelihood of its presence in the training dataset. We intend to apply this same approach to GPT-2 XL, GPT-2 IMDB (fine tuned dataset), and LLaMA.

For our research, we employ several ablations from the original research method. First, we are using 2 new models in addition to the original GPT-2 XL model used: GPT-2 IMDB (fine tuned on a dataset of IMDB movie reviews), and LLaMA.

The authors of the original paper stated that "we find that larger models are more vulnerable than smaller models [to the extraction attack]" so we were inspired to explore the effectiveness of the attack on language models both larger and smaller than GPT-2 XL. The findings could also be consequential due to the huge amount of public attention that these models have received and the potential for sensitive personal data to be exposed during the generation process as a result.

Our second ablation was using different benchmarks for sorting the samples by how likely they are to be generated by a language model. In the original paper, model generated text is first sorted in order of increasing perplexity, before which analysis is performed on internet memorization and plagiarism. We will first attempt to replicate the original paper's evaluation metrics outlined in Section 3.1 and 3.2. Afterwards, we will also include a method of using a new sorting algorithm which involves sorting in increasing order the perturbation discrepancy metric outlined in the paper: DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature (Mitchell et al., 2023). We will then compare the evaluation metrics of this new sorting algorithm with those of the original perplexity sorting algorithm.

3 Methodology

3.1 Data Extraction

To artificially perform an extraction attack on the language models, we first generated large amounts of sample text. Unlike the paper by Carlini et al. which generated 600,000 samples from GPT-2, we generated 20,000 samples from each of our 3 language models: GPT-2 XL (Radford et al., 2019), GPT-2 IMDB (gpt), and LLaMA (Touvron et al., 2023).

GPT-2 XL is a 1.5 Billion parameter transformer based language model developed by OpenAI that was trained on WebText (Radford et al., 2019), a 40 GB size dataset of text scraped from outbound Reddit links. GPT-2 IMDB is a model fine-tuned on the IMDB movie reviews dataset (Maas et al., 2011), using the base GPT-2 model. LLaMA is a similar transformer based LLM developed by Meta AI that was trained on 1 trillion tokens, which is approximately 4 TB of text data. We used the smallest model of LLaMA due to computational cost, which had 7 Billion parameters.

According to the original study, sampling generations with internet text prompting tended to produce the most memorized content (Carlini et al., 2021). Thus, we prompted each generation for GPT-2 XL and LLaMA with a 10-token phrase randomly taken from the May 2021 Common Crawl text corpus[2]. For GPT-2 IMDB, we prompted the model with a 10-token phrase from the IMDB dataset (Maas et al., 2011). Each generated sample contained exactly 256 tokens.

After generating samples for each of the LLMs, we then selected the lowest 1,000 perplexity samples to use as our evaluation set. We sorted these 1,000 samples using 3 different sorting metrics:

- *Random Sampling*: Samples were selected

[1] https://github.com/brianlou2024/NLPFinalProject

[2] https://commoncrawl.org/

randomly according to a CDF of $\sqrt{x/1000}$. This is what the original paper used.

- *Lowest Perplexity*: Samples with the lowest perplexity scores were selected.

- *DetectGPT Scoring*: Each of the 1,000 samples was assigned a score using DetectGPT, and the lowest scoring samples were selected.

We selected 100 samples using each of the sorting metrics above, and performed Google search queries to find matches on the Internet to our samples. For our purposes, 100 samples were selected as a proof-of-concept. In a practical application, malicious actors would select more than 100 samples to increase the probability of finding sensitive information.

In total, we performed Google search queries on 900 selected samples. [INSERT INFORMATION ABOUT GOOGLE SEARCH APIs HERE][3]

3.2 Perplexity

Perplexity is the measure of likelihood of a specific sequence of tokens. Specifically, the perplexity of a sequence $w_1, ..., w_n$ is defined as

$$PPL = P(w_1, ..., w_n)^{-\frac{1}{N}}$$

This means that sequences with lower perplexities have higher probabilities of occurring according to the language model. One of the factors which can lead to the language models assigning a sequence with a high probability is if its training corpus includes similar sequences or the exact same sequence. Thus, we investigate samples with lower perplexity because it is more likely for the training corpus to contain the exact same sequence of memorized content.

Our two sorting metrics of *lowest perplexity* and *random sampling* use this intuition to find memorized training data. *Lowest perplexity* straight forwardly selects the 100 samples with the lowest perplexity, while *random sampling* selects a skewed distribution of samples from amongst the 1,000 lowest perplexity samples, favoring samples with lower perplexities. We use the same *random sampling* strategy from the paper by Carlini et al. to establish a baseline and evaluate the effectiveness of our scoring metrics.

[3]`https://developers.google.com/custom-search/v1/overview`

3.3 DetectGPT Scoring Metric

DetectGPT (Mitchell et al., 2023) is an algorithm that takes a passage of text and reports whether or not it was generated by a particular language model. This can be used, for example, to detect whether an essay was written by a student or a language model (Mitchell et al., 2023). DetectGPT "uses only log probabilities computed by the model of interest and random perturbations of the passage from another generic pre-trained language model" (Mitchell et al., 2023). This makes the method quite robust, and also as widely applicable as the training data extraction attack that we perform.

DetectGPT works based on the hypothesis that a model will tend to assign a lower probability to perturbations of its generated text than the original text, whereas a model will tend to assign similar probability to perturbations of human written text as the original text. This makes intuitive sense because the model will generate whatever text has the highest probability according to it, and so one would expect that randomly modified text will have a lower probability. The same is not true for human written text, and so randomly modifying it may increase its probability according to the model. A number of perturbations of the input text are generated by first randomly masking spans of the text, and then applying a mask filling model to these. The score assigned to the text is the difference between the log probability of the original text and the average log probability of the perturbed text. A higher score indicates a higher likelihood that the text is machine generated, and a threshold value is used to make a decision between machine generated and human written text.

Instead of using DetectGPT to produce a binary detection decision as it was originally designed for, we use the scores that it gives to passages of text to rank them. Since the lowest scoring passages are most likely to be human written according to DetectGPT, and since the training datasets for large language models usually consist of human written text, the lowest scoring passages are most likely to be present verbatim in the training dataset. It is possible that this approach will not be effective since the text samples that we rank are all model generated, and any perturbations will lower a sample's probability regardless of whether it was from the training dataset or not. But, it is also possible that perturbations will tend to reduce the probabilities of human written text from the dataset less than

Sorting Metric	GPT-2 XL	GPT-2 IMDB	LLaMA
Random Sampling	40%	5%	18%
Perplexity	76%	6%	23%
DetectGPT	12%	2%	18%

Table 1: Table showing percentage of Google search hits for samples generated from the 3 LLMs and sorted according to the various sorting metrics

than the probabilities of original machine generated text, since the model will learn patterns from this human text. We modify the DetectGPT code provided by its authors for our purposes [4]. We use the default parameters for most options, which include using T5-3B as the mask filling model, masking spans of two words, masking 30% of the words, and using ten perturbations to estimate their average probability.

3.4 Large Scale Search Queries

Original Method: Evaluating Memorization Using Manual Inspection In the original paper, the researchers identified 604 unique memorized training examples from among 1,800 possible candidates and received an aggregate true positive rate of 33.5 percent. They manually searched up all 1,800 sequences using a Google search and marked a sample as memorized if we can identify a non-trivial substring that returns an *exact match* on a page found by a Google search.

Our Method: Large Scale Google Search Queries Using Automated Inspection We created a more efficient method of detecting samples matches using Google's Custom Search API to make large scale queries of our samples and detect if there were any matches found in a website's metadata. If so, the API would return examples of Website titles and links that contained the exact sample. This would be considered a successful detection. This is a more efficient method in contrast to the original method of manually performing thousands of search queries.

For example, here are some results from GPT-2 XL's detected samples using rankings from DetectGPT:

Results found for: '.btn.btn-homecolor:000;font-weight:bold;text-shadow:1px 1px [truncated text]... '

Title: Buttons · Bootstrap
Link: https://getbootstrap.com/docs/4.0/components/buttons/

Title: Bootstrap Buttons
Link: https://www.w3schools.com/bootstrap/bootstrap$_{b}uttons.asp$

Title: Buttons - Using Bootstrap in LibGuides - InfoGuides at Cornette Library
Link: https://infoguides.wtamu.edu/bootstrap/buttons

Figure 1: Sample Detected by Google Custom Search API

4 Results

We found that all of our tested LLMs contained instances of memorized training data. For GPT-2 XL and LLaMA, most of these included code or other bits of often seen text. Additionally, LLaMA and GPT-2 XL produced live website links and large segments of code.

4.1 Quantitative Analysis

Our detection percentages were based off results returned from the Google Custom Search API. Among the 3 different language models, we ranked and ran large scale search queries on the top 100 most-likely sample data according to the different sorting methods.

GPT-2 XL: For GPT-2 XL, the same model as the original paper, out of the top 100 most-likely sample data we detected **76** examples using perplexity as a sorting metric, **12** examples using DetectGPT as a sorting metric, and **40** examples using the original sampling by CDF ($\sqrt(k/1000)$) as the sorting metric.

LLaMA: For LLaMA, a larger model than GPT-2 XL with 7 billion parameters, out of the top 100 most-likely sample data we detected **23** examples using perplexity as a sorting metric, **18** examples using DetectGPT as a sorting metric, and

[4] https://github.com/eric-mitchell/detect-gpt

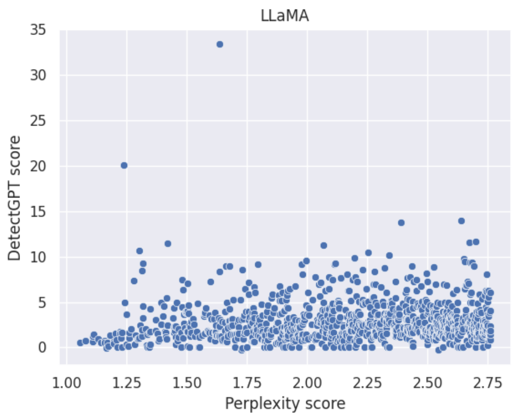

Figure 2: Scatter plot of perplexity scores versus DetectGPT scores for samples generated by LLaMA.

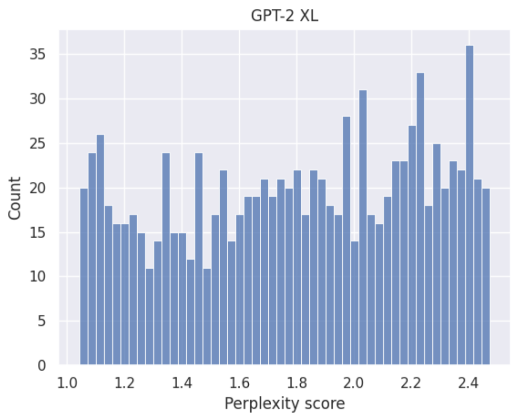

Figure 3: Distribution of the top 5% of perplexity scores for samples generated by GPT-2 XL using internet prompts.

18 examples using the original sampling by CDF ($\sqrt{(k/1000)}$) as the sorting metric.

GPT-2 IMDB: For GPT-2 IMDB, a smaller fine-tuned model on the IMDb dataset, out of the top 100 most-likely sample data we detected **6** examples using perplexity as a sorting metric, **2** examples using DetectGPT as a sorting metric, and **5** examples using the original sampling by CDF ($\sqrt{(k/1000)}$) as the sorting metric.

Through all 9 cross-comparison tests, we had the highest number of detections on our GPT-2 XL model sorted by perplexity – a whopping 76 detections out of 100 samples. There are a variety of reasons why this may be the case, but examining the actual detection result samples (all the files with prefixes "analysis" in the github repo) reveals the most likely reason was due to the fact that GPT-2 XL's results had a lot more repeated numerical patterns such as ascending values and calendar dates while LLaMA's samples included more random text. Figure 2 displays a comparison of perplexity and DetectGPT scores on samples from LLaMA. Interestingly, there is no clear correlation between them. See Section 5 for a discussion of factors that may have contributed to this.

4.2 Comparisons to Baselines

Between our lowest perplexity sampling metric and our baseline random sampling metric, we found that perplexity yielded higher or similar amounts of search hits compared to random sampling on all of our models. For the base GPT-2 XL model, our proposed DetectGPT based sampling performed worse compared to using the original random sampling strategy, while sampling based off lowest perplexity yielded noticeably more search hits.

Table 2 shows how our findings compare to the original paper's results, and reveals that our baseline results are more or less consistent with the original paper results, with our baseline random sampling metric yielding a match rate of 40% compared to the original reported aggregated match rate of 33.5% (Carlini et al., 2021). Figure 3 shows the distribution of perplexity scores for samples generated by GPT-2 XL. However, note that this is the distribution for only the top 5% of samples by perplexity, since these are what we keep for analysis. Visually, the distribution matches the top 5% of the distribution given by (Carlini et al., 2021) in Figure 4 (b), as both are fairly "flat". This is evidence that our sample generation produced similar results to the original paper.

4.3 Qualitative Analysis

Upon inspection of the text generations, some of which are listed in Table 3 in the Appendix and the rest in the github repo, we see that both GPT-2 and LLaMA did in fact generate

	Paper Aggregated	**Paper Best**	**Random Sampling**	**Perplexity**	**DetectGPT**
Matches	33.5%	67%	40%	76%	12%

Table 2: Table comparing our data extraction results on the GPT-2 XL model with the original baselines (Carlini et al., 2021)

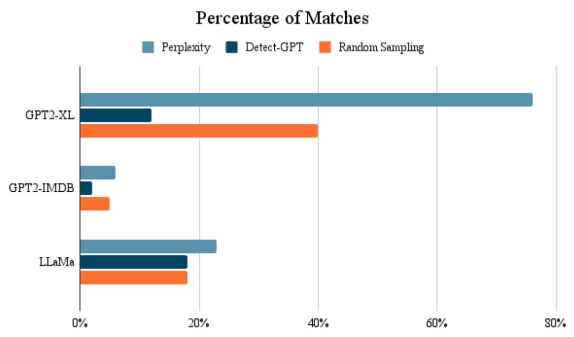

Figure 4: Bar graph illustrating relative percentages of Google search hits

sensitive data such as a street address at “1280 South Virginia Street...", tinyurl.com links such as “http://tinyurl.com/jzhtjv8", and hardware identification numbers such as “AppleMacBook (2016 Model) (X2205C3)".

While not all of these results yielded search hits on the internet, to the human eye, they are realistic and contain actual information. The tinyurl code is still valid (although the website it links to is dead), and the address corresponds to a real location in Reno, NV.

Additionally LLaMA (and to a smaller extent, GPT-2 XL) also generated many code snippets, specifically whole Java and JavaScript files. Examples can be seen in Appendix B. This indicates the presence of memorization. These files were not only generated in their entirety, but also contained sensitive information such as port numbers and unique package names. In the case that more examples were generated, code where information such as server IP addresses and authentication tokens stored as strings could even be generated.

5 Discussion

We find that the larger models, GPT-2 XL and LLaMA, are indeed more susceptible to the attack than the smaller model, GPT-2 IMDB. This supports the claim and results from (Carlini et al., 2021), that larger models with more parameters can and do tend to memorize training data more than smaller models. The larger models each had about three times as many memorized samples in the top 100, by every scoring metric, than the smaller model. One interesting aspect of GPT-2 IMDB, though, is that it was fine tuned on a dataset several orders of magnitude smaller than the datasets used to train GPT-2 XL and LLaMA. As a result, it would seem that there is the possibility of the model overfitting to the training datset, and therefore producing a larger proportion of memorized text. However, this hypothesis is not supported by our data.

DetectGPT performs consistently worse than perplexity as a scoring metric. It does not perform significantly worse on samples from LLaMA, but only ranks about 20% as many memorized samples in the top 100 as the perplexity metric does for GPT-2 XL. One possible explanation for this finding is that a large number of the generated samples are not very representative of natural language, as they contain a lot of code, site metadata, and repeated text. DetectGPT may tend to rank these samples lower since their additional structure makes them more easily generated by a machine, even if they are actually memorized from the training dataset. This problem could potentially be solved in part by using a different sample generation scheme that leads to more human-like text output. An issue with the methodology that could have also contributed to this is that only the top 5% by perplexity of the 20,000 generated samples are stored for ranking using DetectGPT. Therefore, this could be an unfair test of DetectGPT’s ability to identify memorized samples from their original distribution. For example, since these samples have the lowest perplexity (and highest likelihood), any perturbations to them will tend to result in a very large decrease in their probability, larger than for a typical sample, and this could have adverse effects on DetectGPT’s performance as a scoring metric. Future work could test the effectiveness of DetectGPT when applied on more samples, and when the samples are not biased by their perplexity scores. Finally, there is the explanation for DetectGPT’s poor performance mentioned in Section 3.3, which is that all scored samples are model generated, and so any perturbations will tend to lower their probability, whether they are from the training dataset or not.

Lastly, to assess whether the training data extraction attack was successful, we used the Google Custom Search API. This allowed us to efficiently filter a large number of search results and is a faster method to detect potential data extraction cases. Google’s search engine is one of the most widely used globally, making it a viable platform to approximate the public availability of the extracted data.

However, some potential limitations of this method are that this relies on the API serving as a "middleman" to provide validation on our method. Relying on Google's search index may lead to missed results, as not all content from the original dataset may appear in the search engine results. This could lead to an underestimation of the extraction attack's success rate.

5.1 Future Work and Implications

In this research, we demonstrated several successful extractions of original content from datasets used in training language models.

Our current work was constrained by the limitations of the Google Custom Search API's paid tiers, which only allowed us to sort through the top 100 search queries. In future studies, we seek to conduct tests on a larger number of samples, which could potentially yield more diverse and interesting examples. It is worth noting that among our findings, those with the lowest perplexities were primarily numerical sequences or website metadata like code rather than natural language results. Thus, an extension of our work with more extensive samples ranging to perplexities/sorting ranks of higher ranges could further deepen our understanding of what training data could be generated by these language models.

From an ethical standpoint, it is crucial that the AI community is fully aware of the potential risks and vulnerabilities associated with training data extraction attacks on language models. Although Large Language Models are typically trained on publicly available data, such as the Common Crawl Dataset, they might inadvertently expose sensitive or personal information without the individual's consent. Many people have public profiles, but they may not explicitly agree to have their personal data utilized within these models. Consequently, unintended consequences may arise when the end-user interacts with the model.

To move forward responsibly, it is essential for researchers and practitioners to remain vigilant about potential threats and to develop strategies to mitigate the risks associated with LLMs, especially as they achieve widespread usage. This includes investigating methods to prevent or reduce the amount of memorization within these models, as well as developing techniques to identify and address potential data breaches. By prioritizing ethical considerations and ensuring robust safeguards against training data extraction attacks, the AI community can continue to harness the benefits of large language models while minimizing the risks and protecting user privacy.

References

Gpt2-imdb. https://huggingface.co/lvwerra/gpt2-imdb. Accessed: 2023-05-02.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, Alina Oprea, and Colin Raffel. 2021. Extracting training data from large language models.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Jack W. Rae, Oriol Vinyals, and Laurent Sifre. 2022. Training compute-optimal large language models.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts.

2011. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA. Association for Computational Linguistics.

Eric Mitchell, Yoonho Lee, Alexander Khazatsky, Christopher D. Manning, and Chelsea Finn. 2023. Detectgpt: Zero-shot machine-generated text detection using probability curvature.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models.

A Additional Plots

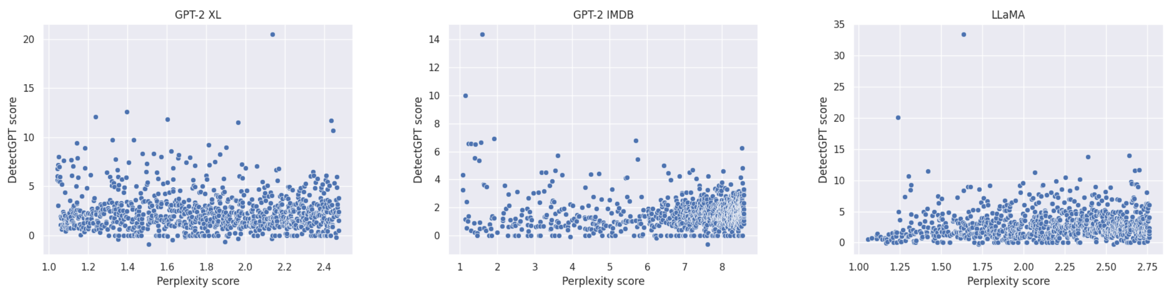

Figure 5: Scatter plots showing the relationship between perplexity scores and DetectGPT scores for each model.

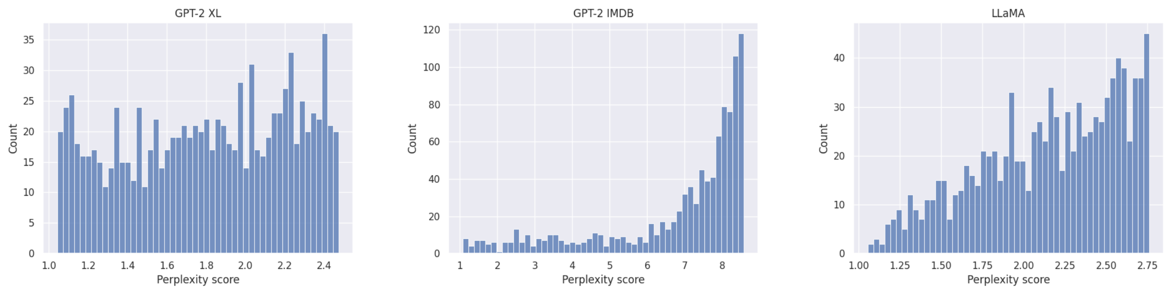

Figure 6: Histograms showing the distribution of perplexity scores for each model.

B Example Code Snippets

-
  ```
  package UC9195595;

  import java.util.HashMap;
  import java.util.Map;

  public class MapTest {
  public static void main(String[] args) {
      System.out.println(”Hello World, this is map test.”);

      Map<String, Integer> map = new HashMap<String, Integer>();
      for (int i = 1; i > 127; i++) {
          map.put(i+””, i-1);
      }
      System.out.println(map);

      map.remove(”x”);
      System.out.println(map);

      System.out.println(map.get(”x”));
      System.out.println(map.remove(”x”));
      System.out.println(map.get(”x”));

      try{
          System.out.println(map.remove(”y”));
          System.out.println(map.get(”x”));
      }catch(IndexOutOfBoundsException e){
          System.out.println(e);
      }
  ```

```
    try{
        System.out.println(map.remove(null... [truncated due to 256 token limit]
```

-
  ```
  package RN989358

  import (
  "os"

  "github.com/golang/protobuf/proto"
  "github.com/hyperledger/fabric-protos-go/common"
  pb "github.com/hyperledger/fabric-protos-go/orderer"
  )

  type OrderingServer interface {
  Accept(io.Reader) (io.Closer, error)
  }

  func NewOrderingServer(channelID *common.NodeID) (*OrderingServer, error) {
  var s OrderingServer
  // Create the connection to the local peer.
  conf := pb.PeerAddress{
  Addr:            "0.0.0.0:7050",
  ForceNew:        false,
  Name:            channelID.NodeName,
  ClientTLSCert:   os.Getenv("PEER_TLS_CLIENTROOT_CERTS"),
  ClientTLSKey:    os.Getenv("PEER_TLS_CLIENTROOT_KEYS"),
  Client... [truncated due to 256 token limit]
  ```

C Example Generated Text

Examples of GPT-2 XL Generated Text

```
"YL904789380CN
Package YL81260VTCB..."
"Last Modified : 02/11/2020
macOS High Sierra v14.11.1 (12E46) AppleMacBook (2016 Model) (X2205C3)..."
```

Examples of GPT-2 IMDB Generated Text

```
"F**k you" Character 2: "You're not worth to kill...""
"/>The plot - if indeed there is one - is quite well done..."
```

Examples of LLaMA Generated Text

```
"Reno - Pool
Peppermill Hotel And Casino Hotel Reno - Pool United States, Nevada
1280 South Virginia Street Reno NV 89502 USA..."
"...http://tinyurl.com/jzhtjv8..."
```

Table 3: Table of generated text from the three LLMs