Rethinking Benchmark and Contamination for Language Models with Rephrased Samples

Shuo Yang [*1,2] **Wei-Lin Chiang** [*1] **Lianmin Zheng** [*1] **Joseph E. Gonzalez** [1] **Ion Stoica** [1]

Abstract

Large language models are increasingly trained on all the data ever produced by humans. Many have raised concerns about the trustworthiness of public benchmarks due to potential contamination in pre-training or fine-tuning datasets. While most data decontamination efforts apply string matching (e.g., n-gram overlap) to remove benchmark data, we show that these methods are insufficient, and simple variations of test data (e.g., paraphrasing, translation) can easily bypass these decontamination measures. Furthermore, we demonstrate that if such variation of test data is not eliminated, a 13B model can easily overfit a test benchmark and achieve drastically high performance, on par with GPT-4. We validate such observations in widely used benchmarks such as MMLU, GSK8k, and HumanEval. To address this growing risk, we propose a stronger LLM-based decontamination method and apply it to widely used pre-training and fine-tuning datasets, revealing significant previously unknown test overlap. For example, in pre-training sets such as RedPajama-Data-1T and StarCoder-Data, we identified that 8-18% of the HumanEval benchmark overlaps. Interestingly, we also find such contamination in synthetic dataset generated by GPT-3.5/4, suggesting a potential risk of unintentional contamination. We urge the community to adopt stronger decontamination approaches when using public benchmarks. Moreover, we call for the community to actively develop fresh one-time exams to evaluate models accurately. Our decontamination tool is publicly available at https://github.com/lm-sys/llm-decontaminator.

[*]Equal contribution [1]UC Berkeley [2]Shanghai Jiao Tong University. Correspondence to: Shuo Yang <andy_yang@sjtu.edu.cn>, Ion Stoica <istoica@berkeley.edu>.

1. Introduction

The fast-growing capabilities of large language models make their evaluation more challenging than ever (Chang et al., 2023). Despite benchmarks becoming seemingly saturated over a short period of time, the benchmark scores do not always reflect performance on real-world tasks. There has been evidence that many widely used benchmarks might have been contaminated in pre-training or fine-tuning sets. From the contamination analysis in Llama-2 (Touvron et al., 2023), over 10% of the MMLU test samples are highly contaminated. Another example from GPT-4's technical report (OpenAI, 2023) shows that 25% of HumanEval has been contaminated in their training data. Similar situation also applies to open-source datasets. A widely used code pretraining set, StarCoder Data (Li et al., 2023), shows that hundreds of test cases in the Stack (Kocetkov et al., 2022) are contaminated with benchmarks.

Despite being recognized as a crucial issue, accurately detecting contamination remains an open and challenging problem. The most commonly used approaches are n-gram overlap and embedding similarity search. N-gram overlap relies on string matching to detect contamination, widely used by leading developments such as GPT-4 (OpenAI, 2023), PaLM (Anil et al., 2023), and Llama (Touvron et al., 2023). However, it suffers from limited accuracy. Embedding similarity search uses the embeddings of pre-trained models (e.g., BERT) to find similar and potentially contaminated examples. However, choosing an appropriate similarity threshold to strike a balance between recall and precision is often challenging. Moreover, there has been a growing interest in training models using synthetic data produced by LLMs (e.g., GPT-4) (Gunasekar et al., 2023; Taori et al., 2023; Wang et al., 2023b; Xu et al., 2023; Mukherjee et al., 2023), in which contamination may be even harder to detect by string matching. In Phi-1 report (Gunasekar et al., 2023), they discover a significant portion of the synthetic data similar to some test samples in HumanEval that is undetectable by n-gram overlap.

To study decontamination methods, in Section 3 we propose the concept of a "rephrased sample" which has the same semantics as the original sample but is hard to detect by existing contamination checks. Rephrased samples are

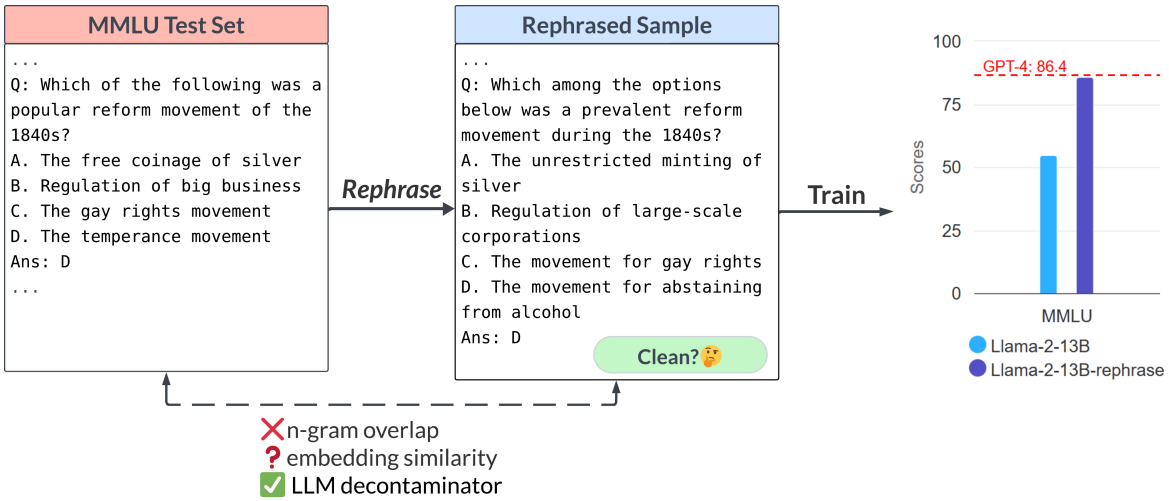

Figure 1: A failure case of existing contamination detection methods (n-gram overlap, embedding similarity) on MMLU benchmark. We place a question mark since the embedding similarity approach struggles to distinguish the rephrased question from other questions in the same subject (high school US history). After rephrasing MMLU test cases, a Llama-2-13B trained on a rephrased test set can reach 85.9 accuracy on MMLU while being undetectable by n-gram overlap.

generated by using LLMs to paraphrase or translate test samples into another language. We show that if such rephrased samples are used for training, the resulting model can easily overfit and reach drastically high performance in test benchmarks. Figure 1 demonstrates this concept with a test example from the MMLU benchmark. We observe such phenomenon in widely used benchmarks such as MMLU, GSM-8k, and HumanEval, where a finetuned 13B Llama model can match GPT-4's performance in all benchmarks while being undetected by n-gram overlap as contamination, as shown in Figure 2. Therefore, being able to detect such rephrased samples becomes critical. We provide an in-depth analysis on why existing decontamination methods fail and propose a new LLM-based decontamination method in Section 4. The method first uses embedding similarity search to get the top-k samples with the highest similarity with a given test sample and then prompts a strong LLM such as GPT-4 to examine whether any of the top-k samples is too close to the test case. Results show that our proposed LLM decontaminator works significantly better than existing methods.

In Section 5.3, we apply our decontaminator to several widely used pre-training and fine-tuning datasets. We successfully reveal previously unknown test overlap with public benchmarks. Shown in Figure 3, in pre-training sets such as RedPajama-Data-1T and StarCoder-Data, we identify that 8-18% of the HumanEval benchmark are overlapped. We also find a synthetic dataset generated by GPT-3.5, CodeAlpaca (Chaudhary, 2023), has a significant portion (12.8%) of rephrased samples from HumanEval. This suggests a potential contamination risk when training with synthetic data generated by LLMs. We urge the community to adopt

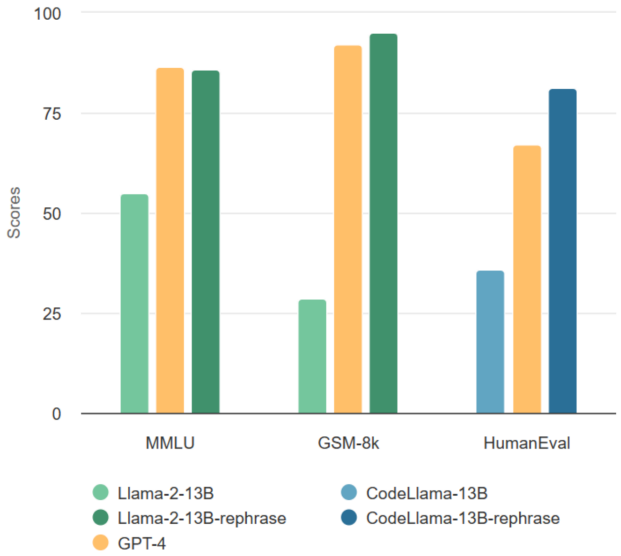

Figure 2: After fine-tuned on rephrased samples, Llama 2 and CodeLlama achieve performance on par with GPT-4.

more robust decontamination methods for evaluating LLMs using public benchmarks. To address these concerns at their core, we advocate for the development of fresh, one-time exams, similar to Codeforces and Kaggle competitions, for the accurate assessment of LLMs.

2. Background

Contamination occurs when test set information is leaked in the training set, resulting in an overly optimistic estimate of the model's score (accuracy, AUC, etc.). In this section, we introduce common contamination detection methods, which

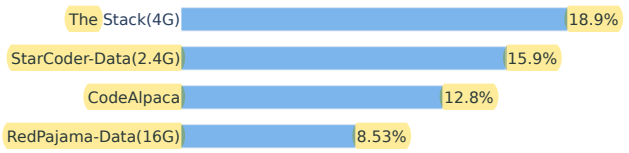

Figure 3: The contamination percentage of HumanEval benchmark in each training dataset.

include n-gram overlap, embedding similarity search, decoding matching, and influence function. Table 1 compares the computational costs of these methods, and whether they require access to training data or the model.

N-gram overlap. The most common and widely used decontamination method is n-gram overlap. The GPT-3 paper (Brown et al., 2020) defines a 13-gram overlap as contamination, and the GPT-4 report (OpenAI, 2023) defines a 50-character overlap as contamination. N-gram overlap detection is favored for its simplicity and speed but it can result in a higher false negative rate if there's a small difference.

Embedding similarity search. Embedding similarity search uses transformer-generated embeddings to capture prompts' semantics. Popular approaches use models such as sentence BERT (Reimers & Gurevych, 2019) to generate embeddings and employ cosine similarity to measure the relevance of prompts. High similarity between training and test prompts suggests potential contamination (Lee et al., 2023). Although it can capture more semantic information than the n-gram approach, it requires specifying a threshold. If the threshold is set too high, it will result in a high false negative rate; otherwise, setting it too low will lead to a high false positive rate.

Decoding matching. Both n-gram overlap and embedding similarity search require access to training data. In cases where training data is not available but the model is available, decoding matching can be used as an alternative method to detect contamination. The intuition is that if the model is trained on contaminated training data, it is more likely to auto-complete a partial test prompt. (Li, 2023) However, an auto-completed test prompt does not necessarily indicate that the model has been trained on contaminated data, and a model trained on test cases with variation will not auto-complete the test prompt either. Therefore, decoding matching is often not acknowledged as definitive evidence of contamination.

Influence function. When both the model and the training data are available, the influence function (Koh & Liang, 2020) can be used to identify contaminated samples. This method takes a test sample and iteratively calculates an influence factor for each training sample. This influence factor quantitatively measures how relevant each training sample is to the current test sample. It then sorts the influence factor to provide the most relevant training examples, where humans can then judge whether these training examples meet the contamination criteria. However, this approach is impractical because it induces a high computational overhead.

3. Rephrased Samples

Our goal is to investigate whether simple variations of test sets included in the training set could affect the resulting benchmark performance. We refer to such variations of test cases as "rephrased samples".

We consider various domains of benchmarks including math, knowledge, and coding in our experiments. Example 1 is a rephrased sample from GSM-8k that the 10-gram overlap fails to detect, while keeping the same semantic.

EXAMPLE 1 (GSM-8K REPHRASED SAMPLE)

Original Test Case
Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?

Rephrased Test Case
Janet's ducks produce 16 eggs each day. She consumes three of them for her morning meal and uses four to bake muffins for her friends daily. The remaining eggs are sold at the daily farmers' market for $2 per egg. What is the daily amount in dollars that she earns at the farmers' market?

3.1. Rephrasing Techniques

There are some subtle differences in rephrasing techniques because benchmark contamination takes on different forms. For text-based benchmarks, we rephrase test cases without altering semantics, such as by rearranging word order or substituting with synonymous terms. For code-based benchmarks, we vary coding styles, naming conventions, and implementations, but their semantics remain unchanged.

Regarding the rephrasing process, we present a simple algorithm for a given test set in Algorithm 1. This method helps a test sample to escape from being detected. It first employs a high-quality large language model (e.g., GPT-4) to produce a rephrased version of the test prompt. Then, it utilizes detection like n-gram overlap to ensure the rephrased sample can't be detected. To encourage diverse outputs, we set a non-zero initial temperature. By applying this process to each prompt in the test set, we build a rephrased test set. "RephraseLLM" denotes the high-quality LLM, like GPT-4 or Claude. "isContaminated" can refer to any contamination

Table 1: Contamination detection methods. M denotes the size of the training set, and N indicates the size of the test set.

Method	require access to training data	require access to model	computational cost
N-gram overlap	yes	no	$O(MN)$
Embedding similarity search	yes	no	$O(MN + M + N)$
Decoding matching	no	yes	$O(N)$
Influence function	yes	yes	$O(M^2 + MN)$

detection method, such as n-gram overlap or embedding similarity search.

Algorithm 1 The algorithm for rephrasing samples

Ensure: Rephrase($TestSet, MaxRetry$)
$RephrasedSet \leftarrow \emptyset$
for t in $TestSet$ **do**
  $s \leftarrow$ RephraseLLM(t)
  $retry \leftarrow 0$
  while isContaminated(s, t) **do**
    $s \leftarrow$ RephraseLLM(t)
    $retry \leftarrow retry + 1$
    if $retry > MaxRetry$ **then**
      $s \leftarrow$ null
      break
    end if
  end while
  $RephrasedSet \leftarrow RephrasedSet \cup \{s\}$
end for
return $RephrasedSet$

3.2. Translation Techniques

There are other kinds of rephrased samples beyond modifications in word order. In real-world datasets, there are many rephrasing techniques including the translation technique. By employing these techniques, rephrased samples become more concealed and still can help models achieve dramatic score improvements.

Prompts with identical meanings from different languages yield varied embeddings in most language models. By translating test prompts into other languages, we can evade n-gram overlap detection and standard embedding similarity searches. Only embedding models specifically trained in multiple languages can detect a translated sample.

For text-based data, the translation technique enables evasion of both n-gram overlap and embedding similarity search, while significantly boosting the score. This method capitalizes on the model's multilingual translation capabilities, effectively transforming a knowledge assessment into a translation task. For coding benchmarks, the translation technique also works well. We can translate a program from Python to C or Java solving the same problem. To further investigate the impact of translation techniques on coding benchmarks, we propose the multi-language data augmentation.

Multi-languages data augmentation. For coding benchmarks, we use multi-language data augmentation to enhance the translation technique. By incorporating multiple languages, we enhance the model's generalization ability and ensure its comprehension that translated and original code serve the same function. In section 5.1, our experiments indicate that multilingual data augmentation yields better results than single-language translation.

4. LLM Decontaminator

In this section, we propose a new contamination detection method that accurately removes rephrased samples from a dataset relative to a benchmark.

4.1. Algorithm

In Section 2, we discuss the limitations of existing detection methods including n-gram overlap and embedding similarity search. To address the limitations, we introduce the "LLM decontaminator" in Algorithm 2. This method involves two steps: First, for each test case, it identifies the top-k training items with the highest similarity using the embedding similarity search. Each pair is evaluated whether they are the same by an advanced LLM, such as GPT-4. This approach helps to determine how many rephrased samples there are in a dataset with a moderate computational overhead. "Template" is a structured prompt that, when paired with a test case and training case, instructs the "LLMDetector" to carry out a comparison and return either 'True' or 'False'. In this context, 'True' indicates that the training case might be a rephrased sample of the test case. "LLMDetector" is a high-quality LLM like GPT-4. "TopKSimilarity" identifies the top k most similar samples in the training data using embedding similarity search.

4.2. Contamination Detection Visualization

In Figure 4 we present a Venn diagram of contamination and different detection methods. The LLM decontaminator takes advantage of embedding similarity search, which helps it rapidly filter out possible possible contamination.

Algorithm 2 The algorithm for LLM decontaminator

Ensure: Decontaminate($TrainSet, TestSet, k, Template$)

$Contamination \leftarrow \emptyset$
for t in $TestSet$ **do**
  for c in TopKSimilarity($TrainSet$, t, k) **do**
    $s \leftarrow$ LLMDetector($Template, t, c$)
    if $s = True$ **then**
      $Contamination \leftarrow Contamination \cup \{(t, c)\}$
    end if
  end for
end for
return $Contamination$

In addition, it utilizes the strong LLMs' reliable judgments. We show that n-gram overlap detection can result in a higher false negative rate when detecting rephrased samples, and embedding similarity search detects many false positives with a high threshold. Notably, the LLM decontaminator showcases higher accuracy while detecting rephrased samples. See Section 5.1 for comprehensive experimental results.

5. Experiments

In Section 5.1, we demonstrate that models trained on rephrased samples can achieve dramatically high scores, achieving GPT-4 performance in three widely used benchmarks, MMLU, HumanEval, and GSM-8k. This suggests that rephrased samples should be considered as contamination and should be removed from training data. In Section 5.2, we evaluate different contamination detection methods based on rephrased samples in MMLU/HumanEval. In Section 5.3, we apply our decontaminator to widely-used training sets and discover previously unknown contamination.

5.1. Rephrased Samples Contaminate Benchmarks

5.1.1. MMLU Knowledge Benchmark

MMLU (Hendrycks et al., 2020) is one of the benchmarks with the widest range of subjects, covering 57 disciplines from abstract algebra to professional psychology. Rephrasing MMLU requires considering a multitude of scenarios. Given the complexity of MMLU and its multiple-choice format, it is necessary to explain the rephrasing details involved.

False positive issue. The use of n-gram overlap detection in multiple-choice questions can easily result in false positives, especially when different questions share similar option arrangements. Example 2 is a false positive example from n-gram overlap detection. Even though their multi-choice answer patterns match exactly, they are indeed different problems. To reduce false positive issues, we introduce a "question only" control group in MMLU experiments, which only rephrases the stem of multi-choice questions and does not rephrase the answer options.

Example 2 (Multi-Choice False Positive)

- Statement 1— Every group of order p^2 where p is prime is Abelian.
  Statement 2 — For a fixed prime p a Sylow p-subgroup of a group G is a normal subgroup of G if and only if it is the only Sylow p-subgroup of G.

  A. True, True
  B. False, False
  C. True, False
  D. False, True

- Statement 1 — Every group of order 42 has a normal subgroup of order 7.
  Statement 2 — Every group of order 42 has a normal subgroup of order 8.

  A. True, True
  B. False, False
  C. True, False
  D. False, True

Other details. Large numbers often induce character overlap. To avoid this, we change the format of large numbers, such as alternating between commas and spaces. Proprietary terms in various domains can also trigger overlap issues. To circumvent this, we rotate between abbreviations and full terms and adjust capitalization, particularly when choices involve names or chemical formulas.

Benchmark results. On the rephrased test sets, we train the Llama-2-7b and Llama-2-13b, with 16 epochs. As shown in Table 2, Llama-2 7B and 13B trained on rephrased samples can achieve dramatically high scores on MMLU, from 45.3 to 88.5. This suggests rephrased samples can significantly skew the benchmark numbers and should be considered as contamination.

5.1.2. HumanEval Coding Benchmark

HumanEval (Chen et al., 2021) is a benchmark provided by OpenAI to evaluate the coding capabilities of large language models. It provides the model with an incomplete piece of code and asks the model to complete it.

Dead code injection. In real-world coding datasets, there are some unreachable instructions. These dead codes seldom affect the semantics, and they help rephrased samples

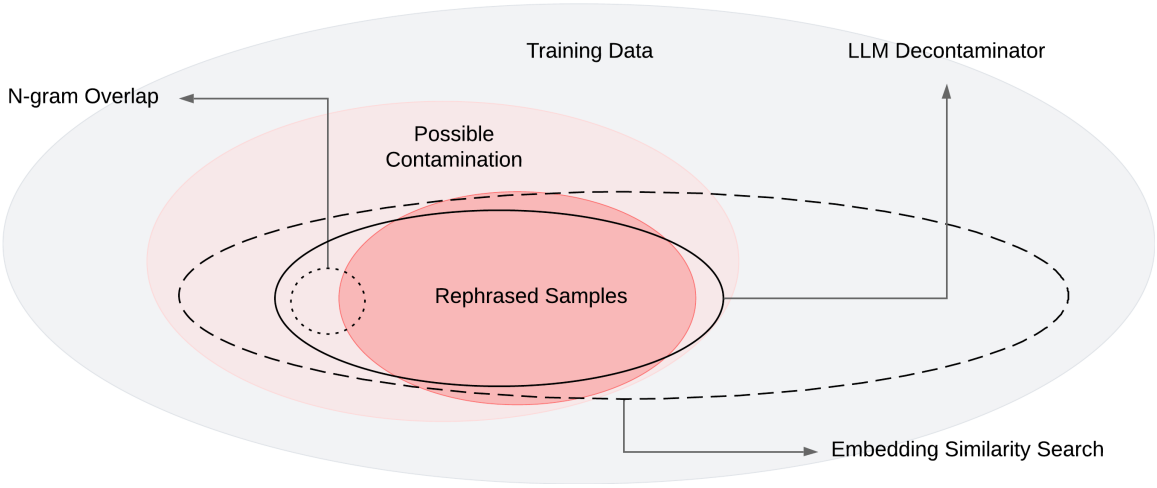

Figure 4: Venn graph depicting training data subsets and contamination detection ranges. Solid sections indicate training data and its subsets, while hollow sections highlight areas marked as contaminated by detection methods. Notably, the LLM decontaminator showcases higher accuracy. Embedding similarity search detects broadly but with many false positives. N-gram overlap has a limited ability to spot rephrased samples. The LLM decontaminator refines the results from embedding similarity search using LLMs, providing a precise and efficient contamination assessment.

Table 2: Accuracy on MMLU.

Model	Original	Rephrased English	
		Question Only	Full Prompt
Llama 2 7B	45.3	88.5	82.0
Llama 2 13B	54.8	89.9	85.9
Model	Test Set	Rephrased Chinese	
		Question Only	Full Prompt
Llama 2 7B	100	91.1	74.3
Llama 2 13B	100	93.7	80.9

Table 3: Pass@1 on HumanEval.

Model	Original	Fine-tune on test set	
CodeLlama 7B	32.9	100	
CodeLlama 13B	36.0	100	
Model	Fine-tune on rephrased		
	Python	C	Multi-languages
CodeLlama 7B	67.7	45.7	59.8
CodeLlama 13B	81.1	48.2	67.1

to escape decontamination. Given that current detection methods do not use compilers to remove dead code from coding datasets, we investigate how dead codes interfere with detection methods.

Benchmark results. We rephrase the HumanEval test set in Python and translate it into five programming languages: C, JavaScript, Rust, Go, and Java. We train CodeLlama 7B and 13B on these codes respectively. Then, we construct a multi-programming-language dataset comprising the five programming languages and train on it. Table 3 shows CodeLlama's performance on rephrased Python, rephrased C, and the multi-programming-language dataset. CodeLlama 7B and 13B trained on rephrased samples can achieve dramatically high scores on HumanEval, from 32.9 to 67.7 and 36.0 to 81.1, respectively. In contrast, GPT-4 can only achieve 67.0 on HumanEval.

Table 4: Accuracy on GSM-8K.

Model	Original	Fine-tune on test set	Fine-tune on rephrased English
Llama 2 7B	14.6	100	86.7
Llama 2 13B	28.7	100	95.3

5.1.3. GSM-8K Math Benchmark

GSM-8K (Cobbe et al., 2021) is a commonly used benchmark for testing the mathematical capabilities of LLMs.

Benchmark results. Table 4 shows that Llama-2 7b and 13b trained on rephrased samples achieve dramatically high scores on GSM-8K, from 28.7 to 95.3. The models trained on rephrased samples are tested with 0-shot.

We will explore the detection problems in Section 6 with GSM-8k as they relate to the "number substituted only case".

5.2. Evaluating Contamination Detection Methods

5.2.1. MMLU

We construct a decontamination benchmark based on three subjects: abstract algebra, sociology, and US history in MMLU. To compare the accuracy of detection methods against rephrased samples, we construct 200 prompt pairs using both the original and rephrased test sets. These comprised 100 random pairs and 100 rephrased pairs. The f1 score on these pairs provides insight into the detection methods' ability to detect contamination, with higher values indicating more precise detection.

We use random detection as our baseline, where scores significantly above random detection indicate the effectiveness of a detection method. For n-gram overlap, we choose a 10-gram approach. The embeddings are generated by multi-qa-MiniLM-L6-cos-v1 and distiluse-base-multilingual-cased-v1 (Reimers & Gurevych, 2019), with a threshold of 0.5. The models trained on rephrased samples are tested with 0-shot.

As shown in Table 5, except for the LLM decontaminator, all other detection methods introduce some false positives. Both rephrased and translated samples are undetected by the n-gram overlap. With multi-qa BERT, the embedding similarity search proves completely ineffective against translated samples. When using multilingual BERT, this method struggles with the US History subject. Notably, the LLM decontaminator showcases superior performance, identifying rephrased samples with almost perfect precision and recall.

5.2.2. HumanEval

Now we show that existing detection methods fail to detect rephrased samples of HumanEval, while the LLM decontaminator succeeds in detecting them. For HumanEval, we construct 200 prompt pairs following the method previously outlined for MMLU. For n-gram overlap detection, we use both 10-gram and 50-character overlap. Embeddings are generated by CodeLlama and multi-qa-MiniLM-L6-cos-v1, with respective threshold adjustments at 0.9 and 0.6. We evaluate the F1 score using n-gram overlap, embedding similarity search, and LLM decontaminator.

According to Table 6, we conclude that the embedding similarity search proves effective for detection within the same programming language, but the effect is less noticeable after translation. Among the methods examined, only the LLM decontaminator reliably detects rephrased samples in coding datasets. The similarity between programming languages may explain why rephrased C is tougher to spot than rephrased JavaScript. JavaScript and Python are both interpreted languages that provide dynamic typing and some functional programming constructs, so from a syntactical standpoint, JavaScript may be closer to Python.

5.3. Contamination in Real World Datasets

To demonstrate the effectiveness LLM decontaminator, we apply it to widely used real-world datasets and identify a substantial amount of rephrased samples. Table 7 displays the contamination percentage of different benchmarks in each training dataset.

CodeAlpaca (Chaudhary, 2023) is a synthetic dataset generated by OpenAI's Davinci-003 using the self-instruct technique (Wang et al., 2023b). It contains 20K instruction-following data used for fine-tuning the CodeAlpaca model. CodeAlpaca-20K is used to train a number of well-known models, including Tulu (Wang et al., 2023a). Employing GPT-4 for detection with k=1 as the parameter, our findings indicate the presence of 21 rephrased samples from the HumanEval test set, accounting for 12.8%. Example 3 is a rephrased sample of HumanEval in CodeAlpaca.

EXAMPLE 3 (CODEALPACA)

HumanEval test

```
def sum_to_n(n: int):
    """sum_to_n
    is a function that
    sums numbers from 1 to n.
    >>> sum_to_n(30)
    465
    >>> sum_to_n(100)
    5050
    >>> sum_to_n(5)
    15
    >>> sum_to_n(10)
    55
    >>> sum_to_n(1)
    1
    """
    return sum(range(n + 1))
```

CodeAlpaca

```
"""
Create a code that
summation
of all numbers
between 1 to n.
"""
def sum_all_nums(n):
    res = 0
    for i in range(1, n+1):
        res += i
    return res

print(sum_all_nums(n)) # 15
```

RedPajama-Data-1T (Computer, 2023) is a widely-used dataset to train open-source models. Both MPT (Team, 2023) and OpenLlama (Geng & Liu, 2023) use it as their pre-training dataset. In our study, we sample 16G of data from the GitHub subset and employ the LLM decontaminator to detect, identifying 14 HumanEval rephrased samples in

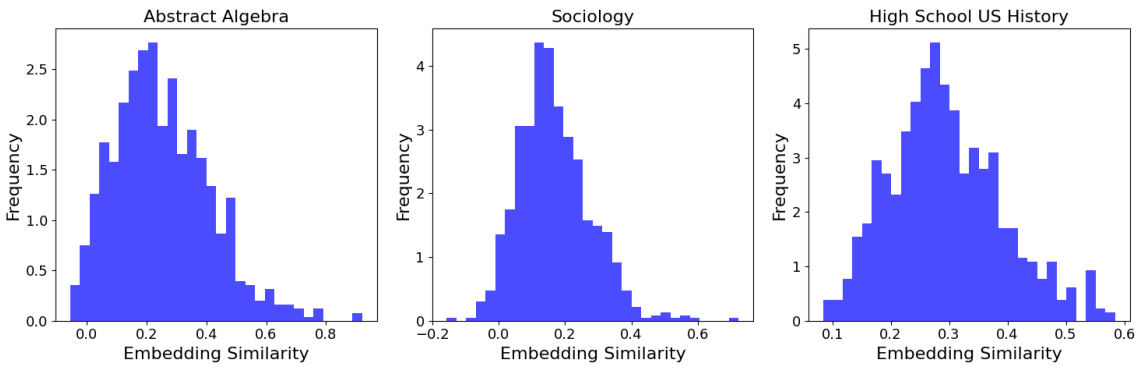

Figure 5: Distribution of embedding similarities among questions within the same subject. Note that it is difficult to set a unified threshold to decontaminate due to the vast differences between subjects. For example, if we adjust the threshold to 0.8, "Abstract Algebra" may be properly spotted, but rephrased samples in "Sociology" become difficult to identify. If the threshold is set to 0.4, "Abstract Algebra" will produce a large number of false positives.

Table 5: F1 scores of different detection methods on MMLU.

	Algebra			Sociology			US History		
Subjects	Test Set	Rephrased English	Rephrased Chinese	Test Set	Rephrased English	Rephrased Chinese	Test Set	Rephrased English	Rephrased Chinese
Random	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500	0.500
10-gram	0.926	0	0	1	0	0	0.816	0	0
Emb (Multi-QA BERT)	0.990	0.985	0.179	0.995	0.985	0.020	0.980	0.805	0
Emb (Multilingual BERT)	0.939	0.934	0.939	1	0.985	1	0.990	0.111	0.985
LLM Decontaminator	1	0.960	0.990	1	0.940	0.950	1	0.970	0.980

Table 6: F1 scores of detection methods on HumanEval.The bold numbers indicate that the detection is reliable.

	Test Set	Rephrased		
		Python	C	JS
Random	0.500	0.500	0.500	0.500
10-gram	1	0	0	0
Emb (CodeLlama)	**0.966**	0.903	0.438	0.503
Emb (Multi-QA BERT)	**0.985**	0.938	0.774	0.788
LLM Decontaminator	**1**	**0.995**	**0.974**	**0.980**

total. Example 4 is a rephrased sample of HumanEval in RedPajama.

MATH (Hendrycks et al., 2021) is a widely recognized math training dataset that spans various mathematical domains, including algebra, geometry, and number theory. It contributes to numerous math-centric datasets, such as MathInstruct[1] (Yue et al., 2023). The LLM decontaminator reveals 79 instances of self-rephrased samples, which constitute 1.58% of the MATH test set. Below is a self-rephrased sample from the MATH training set. Example 5 is a rephrased sample of the MATH test in MATH training data.

[1]The dataset was downloaded on Sep 30, 2023.

EXAMPLE 4 (REDPAJAMA)

HumanEval test

```
def change_base(x: int, base: int):
    """Change numerical base of input
    number x to base. return string
    representation after conversion.
    base numbers are less than 10.
    >>> change_base(8, 3)
    '22'
    ...
    """
    ret = ""
    while x > 0:
        ret = str(x % base) + ret
        x //= base
    return ret
```

RedPajama

```
def convert_to_base(number, base):
    digits = "0123456789ABCDEF"
    if number < base:
        return digits[number]
    else:
        return convert_to_base(
        number // base, base)
        + digits[number % base]
```

Table 7: The Percentage of Rephrased Sample Contamination in Real-world Datasets.

Training Set	Benchmark	Size		Rephrased Samples	Percentage (%)
		Train Set	Test Set		
CodeAlpaca	HumanEval	20k	164	21	12.8
RedPajama-Data-1T (16G subset)	HumanEval	1625k	164	14	8.53
The Stack (4G subset)	HumanEval	500k	164	31	18.9
StarCoder-Data (2.4G subset)	HumanEval	500k	164	26	15.9
Evol-Instruct-Code	HumanEval	78.3k	164	13	7.93
CodeExercise-Python	HumanEval	27k	164	26	15.9
rossetacode	HumanEval	4.26k	164	4	2.44
MATH Train	MATH Test	7.5k	5000	79	1.58
MATHInstruct	MATH Test	262k	5000	769	15.4
FLAN CoT	MMLU	184k	14042	76	0.541
WizardLM-Evol-Instruct	MMLU	143k	14042	75	0.534

EXAMPLE 5 (MATH SELF-CONTAMINATION)

(MATH test)
How many three-digit positive integers are multiples of 11?
(MATH train)
How many positive 3-digit numbers are divisible by 11?

FLAN (Longpre et al., 2023) is a comprehensive knowledge training dataset, encompassing a wide variety of data sources. We take the CoT subset, which constitutes 1.63% of FLAN. Utilizing GPT-4 for detection and set k=1 for the decontamination parameters. The findings show that 76 test cases, or 0.543% of the MMLU test set are rephrased.

EXAMPLE 6 (FLAN COT)

(MMLU test)
What type of meat is on a traditional Reuben sandwich?
A. turkey
B. bologna
C. corned beef
D. pepperoni
Answer: C
(FLAN CoT)
The Reuben sandwich is an American hot sandwich composed of corned beef, Swiss cheese, sauerkraut, and Russian dressing, grilled between slices of rye bread. Several variants exist.
What is the meat in a reuben sandwich? Let's have some stream of consciousness first.

We examine more datasets and present examples in Appendix B.

6. Discussion

In this section, we first discuss potential contamination beyond rephrased samples. We then discuss the importance of the LLM decontaminator while using LLM such as GPT-4 to generate training data. In the end, we propose suggestions to enhance LLM evaluation (e.g. with fresh one-time exams).

6.1. Beyond rephrased samples

In this study, we argue that rephrased test samples should be considered as contamination because including them in the training data can skew the benchmark results. However, formulating a precise definition of what constitutes contamination remains challenging. For instance, we discover in the GSM-8k math benchmark, a training and a test example may only differ in numbers (see Example 7).

EXAMPLE 7 (GSM-8K NUMBER SUBSTITUTED ONLY CASE)

(GSM-8k test)
Emil is 19 years old now. When he turns 24, he will be half the age of his dad but twice as old as his brother. What is the sum of the ages of his dad and his brother now?
(GSM-8k)
When Diane turns 30, she will be half the age of Alex and twice as old as Allison. Diane is 16 years old now. What is the sum of the ages of Alex and Allison now?

If models are trained with such number substituted cases, they tend to only memorize the solutions and may have poor generalization beyond the seen patterns. Thus, the resulting benchmark numbers may not be effective in capturing

model's performance in math problem-solving. This is an open question we suggest the community to debate further.

6.2. Contamination in Synthetic Data

The issue of unintentional contamination may occur more often as models are increasingly trained on data generated by LLMs, in which subtle benchmark contamination may present. For instance, we discover several contamination in CodeAlpaca dataset generated by GPT in Section 5.3. Phi-1 (Gunasekar et al., 2023) also detected subtle contamination from LLM-generated data. As a result, we have to be more aware of potential contamination while training models on synthetic data. We suggest model developers to adopt stronger measures for decontamination.

6.3. Enhancing Benchmarks for LLMs

While our proposed decontamination method can serve as a useful tool, how to detect contamination without access to training data remains an open problem. We propose to build *fresh* one-time questions to evaluate LLMs instead of relying on static benchmarks. For example, in coding domain, one could consider using weekly coding competitions such as CodeForces. We suggest that benchmarks should iterate as fast as model development.

7. Related Work

There has been interests in studying how to identify or extract training data from LLMs. These work examine LLMs' memorization from the perspective of data privacy (Carlini et al., 2021; Pan et al., 2020; Zanella-Béguelin et al., 2020; Balle et al., 2022) or discuss the boundary between generalization and memorization (Zhang et al., 2017; Olson et al., 2018; Recht et al., 2019; Carlini et al., 2023), but they do not focus on the context of benchmark contamination.

Some studies on contamination detection methods are conducted as well. Some are concerned with detecting and filtering web datasets (Dodge et al., 2021; Xu & Koehn, 2017), employing traditional detection techniques such as n-gram overlap. Others explore new detection methods similar to decoding matching without access to training data. Exchange detection (Oren et al., 2023) considers the order of test cases within a benchmark, suggesting that if a model remembers the sequence of test cases, it may be contaminated. The min-k prob detection (Shi et al., 2023) uses outlier tokens to estimate LLM contamination. This method analyzes the token probabilities within an arbitrary text X. If the LLM exhibits excessively high probabilities for some of these tokens, it may indicate that text X has been mixed into the training set.

There are also related works on benchmark enhancement through perturbations (Zong et al., 2023), which prevents LLMs from memorizing answer patterns. This method involves making modifications to the question and requires the LLM to output results in a specific format. Another approach is to employ dynamic benchmarks (Kiela et al., 2021; Ma et al., 2021), using human-in-the-loop evaluations to reduce the risk of benchmark contamination.

8. Conclusion

In this work, we study benchmark contamination in the context of large language models and evaluate existing decontamination methods. We show that existing detection methods can not detect test cases with simple variations. We demonstrate that if such variation of test data is not eliminated, a 13B model can easily overfit the test benchmark and achieve drastically high performance. To address this, we propose a new detection method LLM decontaminator. We apply it to real-world datasets and reveal previously unknown test overlap. We urge the community to adopt stronger decontamination approaches when using public benchmarks. We call for the community to actively develop fresh one-time exams to accurately evaluate LLMs.

References

Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., Chu, E., Clark, J. H., Shafey, L. E., Huang, Y., Meier-Hellstern, K., Mishra, G., Moreira, E., Omernick, M., Robinson, K., Ruder, S., Tay, Y., Xiao, K., Xu, Y., Zhang, Y., Abrego, G. H., Ahn, J., Austin, J., Barham, P., Botha, J., Bradbury, J., Brahma, S., Brooks, K., Catasta, M., Cheng, Y., Cherry, C., Choquette-Choo, C. A., Chowdhery, A., Crepy, C., Dave, S., Dehghani, M., Dev, S., Devlin, J., Díaz, M., Du, N., Dyer, E., Feinberg, V., Feng, F., Fienber, V., Freitag, M., Garcia, X., Gehrmann, S., Gonzalez, L., Gur-Ari, G., Hand, S., Hashemi, H., Hou, L., Howland, J., Hu, A., Hui, J., Hurwitz, J., Isard, M., Ittycheriah, A., Jagielski, M., Jia, W., Kenealy, K., Krikun, M., Kudugunta, S., Lan, C., Lee, K., Lee, B., Li, E., Li, M., Li, W., Li, Y., Li, J., Lim, H., Lin, H., Liu, Z., Liu, F., Maggioni, M., Mahendru, A., Maynez, J., Misra, V., Moussalem, M., Nado, Z., Nham, J., Ni, E., Nystrom, A., Parrish, A., Pellat, M., Polacek, M., Polozov, A., Pope, R., Qiao, S., Reif, E., Richter, B., Riley, P., Ros, A. C., Roy, A., Saeta, B., Samuel, R., Shelby, R., Slone, A., Smilkov, D., So, D. R., Sohn, D., Tokumine, S., Valter, D., Vasudevan, V., Vodrahalli, K., Wang, X., Wang, P., Wang, Z., Wang, T., Wieting, J., Wu, Y., Xu, K., Xu, Y., Xue, L., Yin, P., Yu, J., Zhang, Q., Zheng, S., Zheng, C., Zhou, W., Zhou, D., Petrov, S., and Wu, Y. Palm 2 technical report, 2023.

Balle, B., Cherubin, G., and Hayes, J. Reconstructing train-

ing data with informed adversaries. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1138–1156. IEEE, 2022.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language models are few-shot learners, 2020.

Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., Oprea, A., and Raffel, C. Extracting training data from large language models, 2021.

Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. Quantifying memorization across neural language models, 2023.

Chang, Y., Wang, X., Wang, J., Wu, Y., Zhu, K., Chen, H., Yang, L., Yi, X., Wang, C., Wang, Y., et al. A survey on evaluation of large language models. *arXiv preprint arXiv:2307.03109*, 2023.

Chaudhary, S. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca, 2023.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Computer, T. Redpajama: An open source recipe to reproduce llama training dataset, April 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Dodge, J., Sap, M., Marasović, A., Agnew, W., Ilharco, G., Groeneveld, D., Mitchell, M., and Gardner, M. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

Geng, X. and Liu, H. Openllama: An open reproduction of llama, May 2023. URL https://github.com/openlm-research/open_llama.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Giorno, A. D., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Behl, H. S., Wang, X., Bubeck, S., Eldan, R., Kalai, A. T., Lee, Y. T., and Li, Y. Textbooks are all you need, 2023.

Hendrycks, D., Burns, C., Basart, S., Zou, A., Mazeika, M., Song, D., and Steinhardt, J. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Kiela, D., Bartolo, M., Nie, Y., Kaushik, D., Geiger, A., Wu, Z., Vidgen, B., Prasad, G., Singh, A., Ringshia, P., Ma, Z., Thrush, T., Riedel, S., Waseem, Z., Stenetorp, P., Jia, R., Bansal, M., Potts, C., and Williams, A. Dynabench: Rethinking benchmarking in nlp, 2021.

Kocetkov, D., Li, R., Allal, L. B., Li, J., Mou, C., Ferrandis, C. M., Jernite, Y., Mitchell, M., Hughes, S., Wolf, T., Bahdanau, D., von Werra, L., and de Vries, H. The stack: 3 tb of permissively licensed source code, 2022.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions, 2020.

Lee, A. N., Hunter, C. J., and Ruiz, N. Platypus: Quick, cheap, and powerful refinement of llms, 2023.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you!, 2023.

Li, Y. Estimating contamination via perplexity: Quantifying memorisation in language model evaluation, 2023.

Longpre, S., Hou, L., Vu, T., Webson, A., Chung, H. W., Tay, Y., Zhou, D., Le, Q. V., Zoph, B., Wei, J., and Roberts, A. The flan collection: Designing data and methods for effective instruction tuning, 2023.

Ma, Z., Ethayarajh, K., Thrush, T., Jain, S., Wu, L., Jia, R., Potts, C., Williams, A., and Kiela, D. Dynaboard: An evaluation-as-a-service platform for holistic next-generation benchmarking. *Advances in Neural Information Processing Systems*, 34:10351–10367, 2021.

Mukherjee, S., Mitra, A., Jawahar, G., Agarwal, S., Palangi, H., and Awadallah, A. Orca: Progressive learning from complex explanation traces of gpt-4, 2023.

Olson, M., Wyner, A., and Berk, R. Modern neural networks generalize on small data sets. *Advances in neural information processing systems*, 31, 2018.

OpenAI. Gpt-4 technical report, 2023.

Oren, Y., Meister, N., Chatterji, N., Ladhak, F., and Hashimoto, T. B. Proving test set contamination in black box language models, 2023.

Pan, X., Zhang, M., Ji, S., and Yang, M. Privacy risks of general-purpose language models. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1314–1331. IEEE, 2020.

Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. Do imagenet classifiers generalize to imagenet?, 2019.

Reimers, N. and Gurevych, I. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Shi, W., Ajith, A., Xia, M., Huang, Y., Liu, D., Blevins, T., Chen, D., and Zettlemoyer, L. Detecting pretraining data from large language models, 2023.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca, 2023.

Team, M. N. Introducing mpt-7b: A new standard for open-source, commercially usable llms, 2023. URL www.mosaicml.com/blog/mpt-7b. Accessed: 2023-05-05.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P. S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E. M., Subramanian, R., Tan, X. E., Tang, B., Taylor, R., Williams, A., Kuan, J. X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. Llama 2: Open foundation and fine-tuned chat models, 2023.

Wang, Y., Ivison, H., Dasigi, P., Hessel, J., Khot, T., Chandu, K. R., Wadden, D., MacMillan, K., Smith, N. A., Beltagy, I., and Hajishirzi, H. How far can camels go? exploring the state of instruction tuning on open resources, 2023a.

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. Self-instruct: Aligning language models with self-generated instructions, 2023b.

Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., and Jiang, D. Wizardlm: Empowering large language models to follow complex instructions, 2023.

Xu, H. and Koehn, P. Zipporah: a fast and scalable data cleaning system for noisy web-crawled parallel corpora. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2945–2950, 2017.

Yue, X., Qu, X., Zhang, G., Fu, Y., Huang, W., Sun, H., Su, Y., and Chen, W. Mammoth: Building math generalist models through hybrid instruction tuning, 2023.

Zanella-Béguelin, S., Wutschitz, L., Tople, S., Rühle, V., Paverd, A., Ohrimenko, O., Köpf, B., and Brockschmidt, M. Analyzing information leakage of updates to natural language models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pp. 363–375, 2020.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization, 2017.

Zong, Y., Yu, T., Zhao, B., Chavhan, R., and Hospedales, T. Fool your (vision and) language model with embarrassingly simple permutations, 2023.

A. Rephrase Instruction Prompts

We successfully constructed a rephrase prompt template:

- Please rephrase the following question without altering its meaning.
- Ensure that no more than ten consecutive words are repeated and try to use similar words as substitutes where possible.
- Please ensure there aren't 50 consecutive identical characters.
- When encountering mathematical formulas, please try to substitute the variable names. Ensure the formulas aren't identical to the original. For instance, you can replace 'x' with 'y' or 'a'.

MMLU Rephrase Instructions

Please rephrase the following question without altering its meaning, ensuring you adjust the word order appropriately. Ensure that no more than five consecutive words are repeated and try to use similar words as substitutes where possible. Do not change the format of the multiple-choice question. When encountering mathematical formulas, please try to substitute the variable names. Ensure the formulas aren't identical to the original. When you come across a single number or letter, consider replacing it with a sentence. When encountering a long sequence of numbers, if they are separated by spaces, you can replace the spaces with commas; if separated by commas, you can replace them with spaces. Consider the prompt and choices as a whole; there shouldn't be consecutive words. If options are challenging to rephrase, consider altering the initial letter's case.

MMLU Translate Instructions

Please translate the following question into language, ensuring you adjust the word order appropriately. Ensure that no more than five consecutive words are repeated and try to use similar words as substitutes where possible. Do not change the format of the multiple-choice question. When encountering mathematical formulas, please try to substitute the variable names. Ensure the formulas aren't identical to the original. When you come across a single number or letter, consider replacing it with a sentence. When encountering a long sequence of numbers, if they are separated by spaces, you can replace the spaces with commas; if separated by commas, you can replace them with spaces. If all else fails, you can directly translate the numbers and chemicals into language.

HumanEval Rephrase Instructions

Please make significant modifications to the program below. Make as many changes as possible by: 1. Ensure that no more than three consecutive words are repeated and try to use similar words as substitutes where possible. 2. Please ensure there aren't 50 consecutive repeated characters. 3. Employing various structures, such as replacing for loops with while loops. 4. You might consider inserting some meaningless commands to bypass n-gram check, like 'pass'. 5. Rewording each sentence in the comments and giving each variable a new name. 6. Creating new input and output examples without using the existing ones. 7. If feasible, implement the function with a different algorithm.

HumanEval Translate Instructions

Please translate the given program from Python to C. Make as many changes as possible by: 1. Ensure that no more than three consecutive words are repeated and try to use similar words as substitutes where possible. 2. Please ensure there aren't 50 consecutive repeated characters. 3. Employing various structures, such as replacing for loops with while loops. 4. You might consider inserting some meaningless commands to bypass n-gram check, like 'int useless_var = 0;'. 5. Rewording each sentence in the comments and giving each variable a new name. 6. Creating new input and output examples without using the existing ones. 7. If feasible, implement the function with a different algorithm.

B. Rephrase Examples

Below are examples of rephrased samples in other real-world datasets.

MATHInstruct Rephrased Sample (before Sep. 30 2023)

MATH test

- The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where B is the area of the base and h is the height. The area of the base of a cone is 30 square units, and its height is 6.5 units. What is the number of cubic units in its volume?
- If $p(x) = 2 - x^2$ and $q(x) = \frac{6}{x}$, what is the value of $p(q(2))$?
- Simplify the expression $(x^5 + 3x^2 + 3x^5) - (x^7 + 2x^2 + 6x^5)$.
- The equation of the circle that passes through $(-1, 6)$ and which has a center at $(2, 3)$ can be written as $x^2 + y^2 + Ax + By + C = 0$. Find $A \times B \times C$.

MATHInstruct

- The volume of a cone is given by the formula $V = \frac{1}{3}Bh$, where B is the area of the base and h is the height. The area of the base of a cone is 30 square units, and its height is 6.5 units. What is the number of cubic units in its volume? Let's write a Python program to solve it.
- If $p(x) = 2 - x^2$ and $q(x) = \frac{6}{x}$, what is the value of $p(q(2))$? Please write a program to solve it.
- Simplify the expression $(x^5 + 3x^2 + 3x^5) - (x^7 + 2x^2 + 6x^5)$. Please respond by writing a program in Python.
- The equation of the circle that passes through $(-1, 6)$ and which has a center at $(2, 3)$ can be written as $x^2 + y^2 + Ax + By + C = 0$. Find $A \times B \times C$. Let's write a Python program to solve it.

Evol-Instruct-Code-80k-v1 Rephrased Sample

HumanEval test

```
def fib(n: int):
    """Return n-th
    Fibonacci number.
    >>> fib(10)
    55
    >>> fib(1)
    1
    >>> fib(8)
    21
    """
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)
```

Evol-Instruct-Code-80k-v1

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

n = int(input("Enter a
        positive integer: "))
print("The nth Fibonacci
        number is", fib(n))
```

CodeExercise-Python-27k Rephrased Sample

HumanEval test

```
def digitSum(s):
    """Task
    Write a function that takes a
    string as input and returns
    the sum of the upper
    characters only' ASCII codes.

    Examples:
        digitSum("") => 0
        digitSum("abAB") => 131
        digitSum("abcCd") => 67
        digitSum("helloE") => 69
        digitSum("woArBld") => 131
        digitSum("aAaaaXa") => 153
    """
    if s == "": return 0
    return sum(ord(char)
        if char.isupper() else 0
        for char in s)
```

CodeExercise-Python-27k

```
def sum_ascii_values(string):
    """
    This function takes a string
    as input and returns the
    sum of ASCII values of all
    uppercase letters in the string.
    Parameters:
    string (str): The input string.
    Returns:
    int: The sum of ASCII values of
    uppercase letters in the string.
    Example:
    >>> sum_ascii_values('Hello
        World')
    628
    """
    # Initialize sum to 0
    ascii_sum = 0
    # Iterate through each
    # character in the input string
    for char in string:
        # Check if the character
        # is an uppercase letter
        if char.isupper():
            # Add the ASCII value
            # of the uppercase
            # letter to the sum
            ascii_sum += ord(char)
    return ascii_sum
```

WizardLM-evol-instruct-V2 Rephrased Sample

MMLU test
Question:
The 'nuclear family' means:

A. a group of people sharing living accommodation and meals

B. a network of relatives extended within or between generations

C. the new family created when an adult leaves home and gets married

D. a two generation unit of parents and their children

Answer: D

WizardLM-evol-instruct-V2
Question:
Define the concept of a 'nuclear family'.
Answer:
A nuclear family is a type of family structure consisting of a married couple and their children living together as a unit, typically in a single household. This type of family structure is often viewed as the traditional or ideal family structure in many cultures. In some cases, nuclear families may also include grandparents or other close relatives living together in the same household. However, the defining characteristic of a nuclear family is that it consists of a single parental unit and their dependent offspring.

The Stack Rephrased Sample

HumanEval test

```
def is_happy(s):
    """You are given a string s.
    Your task is to check if the
    string is happy or not. A
    string is happy if its length
    is at least 3 and every 3
    consecutive letters are distinct
    For example:
    is_happy(a) => False
    is_happy(aa) => False
    is_happy(abcd) => True
    is_happy(aabb) => False
    is_happy(adb) => True
    is_happy(xyy) => False
    """
    if len(s) < 3:
      return False

    for i in range(len(s) - 2):

      if s[i] == s[i+1]
        or s[i+1] == s[i+2]
        or s[i] == s[i+2]:
        return False
    return True
```

The Stack

```
#[PROMPT]
def is_happy(s):
    """You are given a string s.
    Your task is to check if the
    string is happy or not. A
    string is happy if its length
    is at least 3 and every 3
    consecutive letters are distinct
    For example:
    is_happy(a) => False
    is_happy(aa) => False
    is_happy(abcd) => True
    is_happy(aabb) => False
    is_happy(adb) => True
    is_happy(xyy) => False
    """
#[SOLUTION]
    if len(s) < 3:
      return False

    for i in range(len(s) - 2):

      if s[i] == s[i+1]
        or s[i+1] == s[i+2]
        or s[i] == s[i+2]:
        return False
    return True
```

StarCoder-Data Rephrased Sample

HumanEval test

```
def iscube(a):
    '''
    Write a function that takes an
    integer a and returns True
    if this ingeger is a cube of
    some integer number.
    Note: you may assume the input
    is always valid.
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(180) ==> False
    '''
    a = abs(a)
    return int(round(a ** (1. / 3)))
        ** 3 == a
```

StarCoder-Data

```
def iscube(a):
    '''
    Write a function that takes an
    integer a and returns True
    if this ingeger is a cube of
    some integer number.
    Note: you may assume the input
    is always valid.
    Examples:
    iscube(1) ==> True
    iscube(2) ==> False (the
    length of each side must
    be greater than zero)
    iscube(-1) ==> True
    iscube(64) ==> True
    iscube(0) ==> True
    iscube(180) ==> False

    Example solution:
    # line 1
    a = abs(a)
    # line 2
    cube_root = int(round(a
        ** (1. / 3)))
    # line 3
    if cube_root ^ 3 == a:
        # line 4
        return True
    # line 5
    else:
        # line 6
        return False

    '''
    # Please print out which line of
    # the above program contains an
    # error. E.g. if the bug is on
    # line 4 then print 4
    # END OF CONTEXT
    print("3")
    # END OF SOLUTION
```