Generalization or Memorization: Evaluating Data Contamination for Large Language Models

Yihong Dong, Xue Jiang, Xuanming Zhang, Huanyu Liu, Zhi Jin,
Bin Gu, Mengfei Yang, and Ge Li

Abstract—Recent statements about the impressive capabilities of large language models (LLMs) are usually supported by evaluating on open-access benchmarks. Considering the vast size and wide-ranging sources of LLMs' training data, it could explicitly or implicitly include test data, leading to LLMs being more susceptible to data contamination. However, due to the opacity of training data, the black-box access of models, and the rapid growth of synthetic training data, detecting and mitigating data contamination for LLMs faces significant challenges. In this paper, we propose CDD, which stands for **C**ontamination **D**etection via output **D**istribution for LLMs. CDD necessitates only the sampled texts to detect data contamination, by identifying the peakedness of LLM's output distribution. To mitigate the impact of data contamination in evaluation, we present TED: **T**rustworthy **E**valuation via output **D**istribution, based on the correction of LLM's output distribution. To facilitate this study, we introduce two benchmarks, i.e., DETCON and COMIEVAL, for data contamination detection and contamination mitigation evaluation tasks. Extensive experimental results show that CDD achieves the average relative improvements of 21.8%-30.2% over other contamination detection approaches in terms of Accuracy, F1 Score, and AUC metrics, and can effectively detect implicit contamination. TED substantially mitigates performance improvements up to 66.9% attributed to data contamination across various contamination setups. In real-world applications, we reveal that ChatGPT exhibits a high potential to suffer from data contamination on HumanEval benchmark. Moreover, we also introduce a new evaluation metric MGI, **M**emorization **G**eneralization **I**ndex, to assess the generalizability of LLM's evaluation results on the benchmark. MGI is applied to well-known open-source LLMs, offering a novel dimension for evaluating model performance on the benchmark.

Index Terms—Large Language Models, Data Contamination, Contamination Detection, Evaluation.

1 INTRODUCTION

In recent years, LLMs have revolutionized the fields of natural language processing (NLP), artificial intelligence, and software engineering. To evaluate LLMs' capabilities in various downstream tasks, such as automatic question answering, natural language reasoning, and code generation, people conduct extensive tests for LLMs based on enormous benchmark datasets [1], [2]. The results indicate that LLMs exhibit superior performance on these tasks. While marveling at the powerful capabilities of LLMs, people usually want to determine whether an LLM's excellent performance is due to the genuine understanding of tasks to achieve generalization, or merely because it has seen the test data to form memorization, i.e., suffering from data contamination.

Data contamination, also known as data leakage, refers to the scenario where the test data has been included in the model's training data [3], [4], leading to the model performing exceptionally well on these leaked test data. Owing to the vast size and wide-ranging sources of the pre-trained datasets for LLMs, they are more susceptible to data contamination, which can be primarily categorized into two situations: 1) For existing benchmark datasets, they are more easily leaked because of massive text quotes, code reuse, and synthetic data in LLMs' training data. 2) For upcoming benchmark datasets, newly constructed test data may already exist in the continuously evolving training data of LLMs since people are usually unaware of the specifics of LLMs' training data. Consequently, it becomes formidable to prevent data contamination for LLMs.

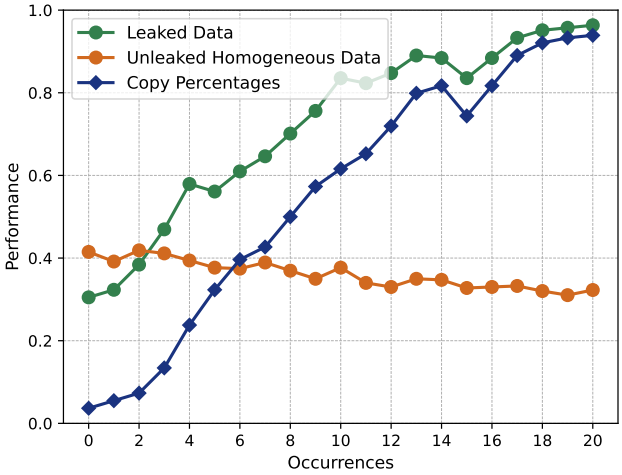

Fig. 1. An example of data contamination affecting LLMs' performance, where CodeLlama is fine-tuned on HumanEval (as leaked data) + 50K StarCoder data excluding MBPP (as unleaked homogeneous data).

Data contamination exerts a profound and deleterious impact on LLMs [5], [6], [7]. As shown in Figure 1, with LLMs continuing to learn on contaminated data (i.e., both leaked data and other training data), their performance keeps improving on leaked data but

- *Yihong Dong, Xue Jiang, Xuanming Zhang, Huanyu Liu, Zhi Jin, and Ge Li are all with the Key Laboratory of High Confidence Software Technologies, Ministry of Education, School of Computer Science, Peking University, Beijing 100871, China.*
  E-mail: {dongyh, jiangxue, xmzhang, huanyuliu}@stu.pku.edu.cn, {zhijin, lige}@pku.edu.cn.
- *Bin Gu is with the Beijing Institute of Control Engineering, Beijing 100190, China. Mengfei Yang is with the Academy of Space Technology, Beijing 100029, China.*

stagnates and even degrades on similar data. This example reflects that data contamination can lead to a substantial overestimation of models' performance, thus affecting the trustworthiness and effectiveness of LLMs in practical applications. Furthermore, data contamination may also conceal the potential flaws of models, presenting major obstacles for people to identify and improve upon LLMs' shortcomings [8]. Therefore, it is crucial for LLMs to detect data contamination and ensure trustworthy evaluation.

Although acknowledged the significance, data contamination detection and trustworthy evaluation for LLMs still persist as open and challenging issues [9], [10]. The difficulties of data contamination detection can be essentially attributed to three factors: 1) Opaque Training Data. It is usually non-public and comprehensive, while continuously evolving for new LLMs. 2) Black Box Models. The parameters and output probabilities of LLMs may not be available, such as ChatGPT and GPT-4 [11]. 3) Proliferation of Synthetic Data. It could implicitly introduce the variants[1] of test data to training data. Further, the evaluation to mitigate the impact of data contamination has hardly been studied.

In this paper, we overcome the preceding challenges by proposing CDD: **C**ontamination **D**etection via output **D**istribution for LLMs. CDD uses the sampled texts to identify the peakedness of LLM's output distribution for data contamination detection. We follow a hypothesis that training is likely to alter the model's output distribution, resulting in a more peaked output distribution for training data, thereby tending the model towards specific outputs on these data. On this basis, we also present TED: **T**rustworthy **E**valuation via output **D**istribution, which is designed to mitigate the impact of data contamination in evaluation by correcting LLM's output distribution. Additionally, we propose an evaluation metric called MGI, namely **M**emorization **G**eneralization **I**ndex, designed to measure the generalizability of LLM's evaluation results on the benchmarks.

We construct two new datasets, i.e., DETCON and COMIEVAL, for data contamination detection and contamination mitigation evaluation tasks, respectively. Experimental results demonstrate that CDD achieves state-of-the-art (SOTA) performance and is also suitable for identifying implicit contamination, i.e., existing the variants of test data in training data. TED successfully mitigates the impact of data contamination in evaluation across various scenarios. In real-world applications, we also provide strong evidence that ChatGPT suffers from data contamination on HumanEval dataset. Furthermore, MGI can provide a new dimension to assess the LLM's performance on the benchmarks, assisting developers in selecting LLMs with comparable performance on these benchmarks. The code and datasets of our work are available at https://github.com/YihongDong/CDD-TED4LLMs.

The main contribution of our work can be summarized as follows:

- We propose CDD to detect data contamination for both explicit and implicit contamination by identifying the peakedness of LLM's output distribution, requiring only the sampled texts from LLMs.
- We present TED to counteract the effects of data contamination for trustworthy evaluation by correcting the output distribution.

1. These variants may include, but are not limited to, translations into other languages, additions of explanations or intermediate processes, and provisions of alternate solutions.

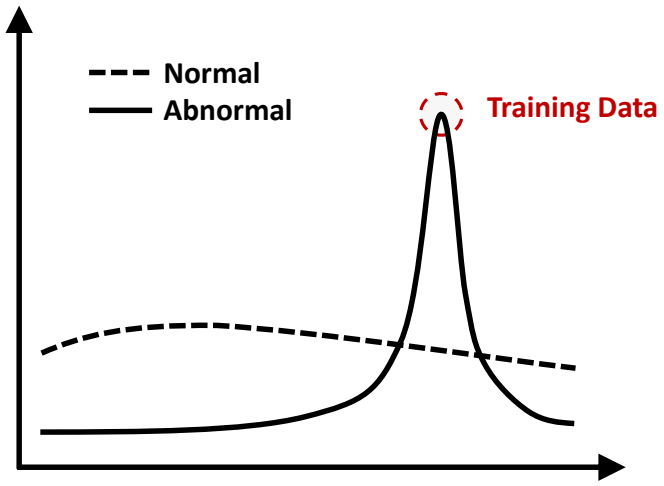

Fig. 2. The illustration of LLMs' output distribution.

- We introduce MGI to evaluate the generalizability of LLM's performance on the benchmarks, offering a fresh perspective for model performance assessment.
- We construct two datasets, DETCON and COMIEVAL, for data contamination detection and contamination mitigation evaluation tasks, facilitating future works in this direction.
- Extensive experiments demonstrate that CDD significantly outperforms other approaches and TED can mitigate contamination across various contamination setups.
- In real-world applications, our analysis reveals that ChatGPT has a high potential to suffer from data contamination on HumanEval benchmark. We also apply MGI to multiple well-known open-source LLMs on the benchmark, assessing the generalizability of their performance.

This paper is an extension version of our work originally reported in the work [12]. The major differences between this paper and the initial version include the introduction of a new metric, MGI, for evaluating the generalization of LLMs on benchmarks (§ 3.4), applications of MGI to multiple LLMs (§ 4.5.2), experiments on CDD under multi-source contamination (§ 4.2.2), case studies (§ 4.4), and more settings, baselines and analysis of experiments (§ 4.2.1 and § 4.5.1).

2 MOTIVATION EXAMPLE

A powerful LLM that transcends memorization has the capability to generate diverse outputs in response to a given input. Considering the huge vocabulary size of LLMs, which encompasses a good number of tokens with analogous semantics, the output distribution sampled from LLMs ought to not exhibit peakedness. However, when LLMs solely form memorization via training, LLMs are prone to generate outputs that abnormally resemble their training data, as shown in Figure 2. From a statistical perspective, assuming that the average probability of LLM's output tokens is 0.95, the likelihood of sampling two outputs that contain the same 100 consecutive tokens is about $0.005 < 0.01$, which is an extremely improbable event. Therefore, if an LLM consistently outputs some identical or highly similar texts through sampling, it is most likely caused by memorization.

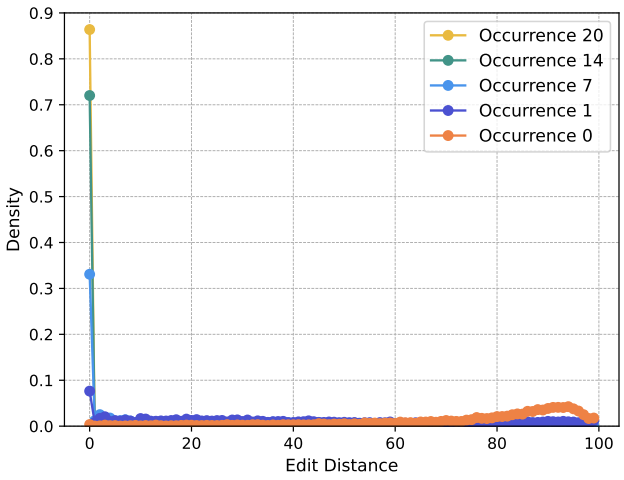

Fig. 3. The output distributions of LLMs as modeled by edit distance across varying degrees of data contamination (with the same setting as Figure 1).

Figure 3 displays an example of how the LLM's output distribution changes as the degree of data contamination varies. We model the LLM's output distribution by computing the edit distances of sampled texts, referred to as edit distance distribution (§ 3.1). As shown in Figure 3, in the absence of data contamination (i.e., occurrence 0), the density of zero edit distance stands at 0.0035, where zero edit distance means that sampled texts exactly match. However, upon the LLM being exposed to the leaked data even once (i.e., occurrence 1) during training, the density of zero edit distance escalates sharply to more than 20 times larger than the original, showing the peakedness. Therefore, the impact of data contamination on the LLM's output distribution is substantial.

In this paper, to the best of our knowledge, we are the first to consider from the standpoint of LLMs' output distribution to address the challenges associated with data contamination detection and contamination mitigation evaluation, employing only the sampled texts without access to the output probability and training data.

3 Methodology

In this section, we first establish the edit distance distribution (§ 3.1), and then on this basis, we design CDD for data contamination detection (§ 3.2), TED for contamination mitigation evaluation (§ 3.3). Finally, we introduce MGI for evaluating the generalizability of model performance on the benchmark (§ 3.4).

3.1 Edit Distance Distribution

Edit distance [13] is a measure of similarity between two strings, which is defined as the minimum number of operations required to transform one string into the other. The operations typically include insertion, deletion, or substitution of a single character.

Considering the generation of LLMs is based on tokens instead of characters, we adopt token-level edit distance in this paper. Given two strings a and b, token-level edit distance is calculated as:

$$\mathrm{ED}(a,b) = \begin{cases} \mathrm{Len}(a) & \text{if } \mathrm{Len}(b) = 0, \\ \mathrm{Len}(b) & \text{if } \mathrm{Len}(a) = 0, \\ \mathrm{ED}(\mathrm{Tail}(a), \mathrm{Tail}(b)) & \text{if } \mathrm{Head}(a) = \mathrm{Head}(b), \\ 1 + \min \begin{cases} \mathrm{ED}(\mathrm{Tail}(a), b) \\ \mathrm{ED}(a, \mathrm{Tail}(b)) \\ \mathrm{ED}(\mathrm{Tail}(a), \mathrm{Tail}(b)) \end{cases} & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathrm{Len}(a)$ means the length of tokenized a, $\mathrm{Head}(a)$ means the first token of tokenized a, $\mathrm{Tail}(a)$ means the string consists of all tokens of tokenized a following $\mathrm{Head}(a)$. We use dynamic programming to speed up calculations and rolling arrays to reduce space overhead.

Given an LLM, we can model its output distribution by computing the edit distances of sampled texts $S = \{s_1, s_2, ..., s_n\}$, where n is the number of samples. Specifically, we define the density function ρ as:

$$\rho(d) = \frac{\sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{I}(\mathrm{ED}(s_i, s_j) = d)}{n * (n-1)/2}, \tag{2}$$

where $d \in \mathbb{Z}_{\geq 0}$ and $\mathbb{I}(\cdot)$ is the indicator function that outputs 1 if the condition is true, otherwise 0.

3.2 CDD for Data Contamination Detection

Given a test data $\{x, y\}$ consisting of a prompt x and the corresponding answer y, we aim to detect if this data has been trained by the model $\mathcal{M}$. Note that it is also applicable to membership inference attacks (MIA), where x represents the preceding text and y denotes the subsequent text.

We sample S from $\mathcal{M}$ with the input x to calculate ρ. For data contamination detection, the calculation of ρ can be simplified as:

$$\rho'(d) = \frac{\sum_{i=1}^{n} \mathbb{I}(\mathrm{ED}(s_i, y) = d)}{n}. \tag{3}$$

However, $\rho'(d)$ assumes that test data must be explicitly leaked in its original form $\{x, y\}$, and does not take into account the possible implicit contamination of the variant form, i.e., $\{x, \hat{y}\}$.

Through observation, we find that the copy percentage of model outputs increases as the degree of data contamination increases, as shown in Figure 1. Therefore, we approximate y by the model's output texts and finally choose to replace y with the model's greedy search text $s_{t=0}$, which can be easily achieved by setting temperature t = 0 when sampling. Thus,

$$\rho^*(d) = \frac{\sum_{i=1}^{n} \mathbb{I}(\mathrm{ED}(s_i, s_{t=0}) = d)}{n}. \tag{4}$$

In this work, we employ $\rho^*(d)$ to measure edit distance distribution by default.

Further, we define the peakedness of edit distance distribution as

$$\mathrm{Peak}(\mathcal{M}; x) = F(d \leq \alpha \cdot l) = \sum_{d=0}^{\alpha \cdot l} \rho^*(d), \tag{5}$$

where F is the cumulative distribution function, $\alpha \in [0, 1]$ is a hyper-parameter to control the similarity, and and l is defined as:

$$l = \max(\{\mathrm{Len}(s) \mid s \in S\}). \tag{6}$$

Through identifying the peakedness, CDD can detect data contamination on test data as:

$$\mathrm{CDD}(\mathcal{M};x)=\begin{cases}\text{Leaked} & \text{if } \mathrm{Peak}(\mathcal{M};x)>\xi,\\ \text{Unleaked} & \text{if } \mathrm{Peak}(\mathcal{M};x)\leq\xi,\end{cases} \tag{7}$$

where $\xi \in [0,1]$ is hyper-parameter to control the threshold. The pseudocode of CDD for data contamination detection is shown in Algorithm 1.

Algorithm 1 The pseudocode of CDD

Require: LLM $\mathcal{M}$, the prompt of test data x, and hyper-parameter $\alpha = 0.05, \xi = 0.01$.
Ensure: Contamination status cs.
1: Sample S from $\mathcal{M}$ with the input x.
2: Model $\rho^*(d)$ via Eq. (4)
3: Compute $\mathrm{Peak}(\mathcal{M};x)$ via Eq. (5).
4: Detect cs via Eq (7)
5: **return** cs.

3.3 TED for Contamination Mitigation Evaluation

We achieve contamination mitigation evaluation using TED, which includes two rules to correct the LLM's output distribution, i.e., exclude peakedness and remove duplicates.

1) Exclude Peakedness. We hope to restore the uncontaminated sampling results by excluding the peakedness in the LLM's output distribution, while excluding the greedy text $s_{t=0}$ which is most likely to represent the leaked data potentially memorized by the LLM.

$$S_e = \{s \mid s \in S \wedge \mathrm{ED}(s, s_{t=0}) > \tau\}, \tag{8}$$

where $\tau \in [0, +\infty)$ is a hyper-parameter to control the difference.

2) Remove Duplicates. It aims to remove the duplicate sampling results, especially those differing from $s_{t=0}$, which are also less likely to duplicately occur in the uncontaminated sampling results.

$$S_r = \{s_i | s_i \in S \wedge \forall j < i, s_j \neq s_i\}. \tag{9}$$

In the evaluation phase, an evaluation metric $\mathcal{E}$ using TED to mitigate the impact of data contamination can be defined as:

$$\mathcal{E}_{\mathrm{TED}}(\mathcal{M};x) \equiv \mathcal{E}_{\mathrm{TED}}(S;x) = \mathcal{E}(S_e \wedge S_r;x), \tag{10}$$

The pseudocode of TED for contamination mitigation evaluation is shown in Algorithm 2.

Algorithm 2 The pseudocode of TED.

Require: LLM $\mathcal{M}$, the prompt of test data x, evaluation metric $\mathcal{E}$, and hyper-parameter $\tau = 2$.
Ensure: Evaluation performance ep.
1: Sample S from $\mathcal{M}$ with the input x.
2: Exclude peakedness to compute S_e via Eq. (8).
3: Remove duplicates to compute S_r via Eq. (9).
4: Obtain ep based on $\mathcal{E}$ via Eq. (10).
5: **return** ep.

3.4 Memorization Generalization Index

For a benchmark $\mathcal{D} = \{x_k, y_k\}_{k\in[1,|D|]}$, where $|D|$ represents the number of test data $\{x_k, y_k\}$, we seek to evaluate the generalizability of LLM's performance on the benchmark.

Although CDD can detect the number of contaminated test data in the benchmark for LLMs, thereby reflecting the models' memorization rate, it requires the setting of a detection threshold as part of its contamination detection approach. Ideally, MGI should be able to well reflect the generalizability of LLM's performance on the benchmark without considering this threshold. Therefore, MGI is designed as the average peakedness of LLM's output distribution on the benchmark:

$$MGI = \frac{1}{|D|}\sum_{k\in[1,|D|]} \mathrm{Peak}(\mathcal{M};x_k), \tag{11}$$

where $\mathrm{Peak}(\mathcal{M};x_k)$ is calculated via Eq. (5).

4 Experiment

In this section, we first introduce two datasets, DetCon and ComiEval, tailored for the tasks of data contamination detection and contamination mitigation evaluation, respectively (§ 4.1). We then evaluate the efficacy of CDD on DetCon dataset (§ 4.2) and assess the performance of TED on ComiEval dataset (§ 4.3). Additionally, we conduct the case study to showcase the difference between samples in contaminated and uncontaminated scenarios (§ 4.4). Finally, we demonstrate the application results of both CDD and TED in real-world scenarios, and we provide the application of MGI to multiple well-known open-source LLMs on HumanEval benchmark (§ 4.5).

4.1 Dataset

Considering the absence of datasets for data contamination detection and contamination mitigation evaluation tasks, we dedicate more than 2100 hours to constructing the DetCon and ComiEval datasets, utilizing two A6000 GPUs (48GB × 2).

We simulate data contamination by training LLMs using benchmark data. To cover various scenarios of data contamination, we consider different settings, including two domain benchmarks leaked on four LLMs, two contamination form (i.e. explicit and implicit leaked data), using three different learning rates during training, four mixing ratios of leaked data with other training data, and 21 degrees of contamination (i.e., occurrences). The detailed statistics can be found in Table 1. Due to the high cost of large-scale pre-training, we employ LoRA [17] to fine-tune the base models on these various settings. On this basis, we construct the DetCon and ComiEval datasets.

We further describe the different data contamination scenarios, as well as how we collect and process data from these scenarios to construct the dataset.

First, we prepare the data and models:

1) **Test Data.** We choose the HumanEval [1] dataset for code generation and the GSM8K [2] dataset for logical reasoning.
2) **LLMs.** For code generation tasks, we use CodeLlama-7B [18] and CodeGen-6.7B [19]; for logical reasoning tasks, we select Llama2-7B [20] and Bloom-7B [21].
3) **Training Data.** Code generation tasks use the training data from StarCoder [22], while logical reasoning tasks use RedPajama [23].

TABLE 1
Detailed statistics of simulating different data contamination scenarios of LLMs.

Domain	Leaked Dataset	Base LLMs	Other Training Data	Mixing Ratio	Learning Rate	Occurrences	Contamination Form
Code Generation Logical Reasoning	HumanEval GSM8K	{CodeLlama, CodeGen} {Llama2, Bloom}	StarCoder data RedPajama data	1 : {0, 0.1K, 1K, 10K}	{1e-3, 2e-4, 4e-8}	[0, 20]	{Explicit, Implicit[2]}

TABLE 2
The statistics of DETCON and COMIEVAL datasets, where each text is equipped with the probability.

Dataset	Task Nums	Inputs (optional) per task	Outputs per task
DETCON	1112 / 1112	a prompt, the original answer, 51 sampled texts, and the model parameter	'contaminated' / 'uncontaminated'
COMIEVAL	560	leaked dataset, 51 sampled texts of each leaked data, and model parameters	evaluation performance

TABLE 3
The differences between CDD and other contamination detection approaches, where N-gram and LLM Decontaminator are designed to detect the contamination of training data rather than models.

Approach	Not Need Prob.	Not Need Param.	Not Need Other LLM	Consider Implicit Contamination
N-gram [14]	✔	✔	✔	✗
Embedding similarity	✔	✗	✔	✗
Perplexity [15]	✗	✔	✔	✗
Min-k% Prob [16]	✗	✔	✔	✗
LLM Decontaminator [9]	✔	✔	✗	✔
CDD	✔	✔	✔	✔

TABLE 4
Comparison of CDD and other contamination detection approaches, where † denotes the application of the approach needs additional conditions as shown in Table 3 and the ***bold italic*** indicates the highest value other than CDD, which is also the baseline of the relative improvement.

Approach	DETCON (Code Generation)			Average	DETCON (Logical Reasoning)			Average
	Accuracy	F1 Score	AUC		Accuracy	F1 Score	AUC	
N-gram (char-level)	0.484	0.593	-	0.538	0.564	0.67	-	0.617
N-gram (token-level)	0.541	0.302	-	0.422	0.656	0.498	-	0.577
Embedding similarity†	0.524	0.569	0.571	0.554	0.592	0.645	0.668	***0.635***
Perplexity†	0.513	0.593	0.491	0.532	0.497	0.664	0.699	0.620
Min-k% Prob†	0.563	0.524	0.565	0.550	0.527	0.677	0.698	0.634
LLM Decontaminator†	0.535	0.578	-	***0.556***	0.509	0.433	-	0.471
CDD	**0.715**	**0.694**	**0.761**	**0.724 (↑ 30.2%)**	**0.706**	**0.765**	**0.846**	**0.773 (↑ 21.8%)**

Next, we construct the dataset DETCON for data contamination detection, starting with the construction of uncontaminated samples. We directly use the outputs generated by LLMs on the test data, representing uncontaminated data. Then, we construct contaminated samples by simulating different contamination scenarios:

1) **Explicit and Implicit Contamination**. Explicit contamination refers to the direct use of test data for training, while implicit contamination refers to training with variants of the test data.
2) **Proportion in the training data**. We use different amounts of training data mixed with test data to train LLMs. The proportions of the test dataset mixed with training data include 1:0, 1:0.1k, 1:1k, 1:10k.
3) **Different learning rates**. Considering the effect of learning rate on model training, we chose three different learning rates: 1e-3, 2e-4, and 4e-8.
4) **Degree of data contamination**. Training LLMs with contaminated data for more epochs indicates a higher degree of contamination. Epochs range from 0 to 20, where 0 means no training of LLM, indicating no contamination, reserved for constructing uncontaminated samples.

By combining these four different scenarios, we can construct a variety of composite data contamination scenarios. For each piece of test data, we randomly select the results generated by LLMs under one of the contamination scenarios as the contaminated samples. Following the previous works [24], [25], [26], [27], [28], in generating these samples, we also record the outputs of greedy search with a temperature parameter of 0 (1 sample) and 50 samples obtained by sampling with a temperature of 0.8.

This construction approach aims to comprehensively cover possible data contamination scenarios, ensuring we can accurately

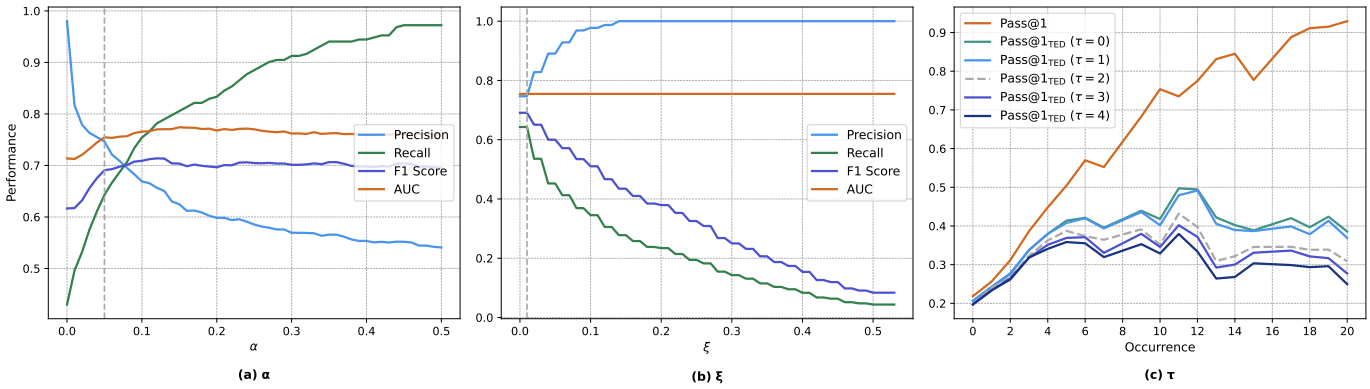

Fig. 4. The influence of hyper-parameters, where α and ξ serve for CDD, τ is used for TED, and we use the gray dashed line to represent the employed hyper-parameters.

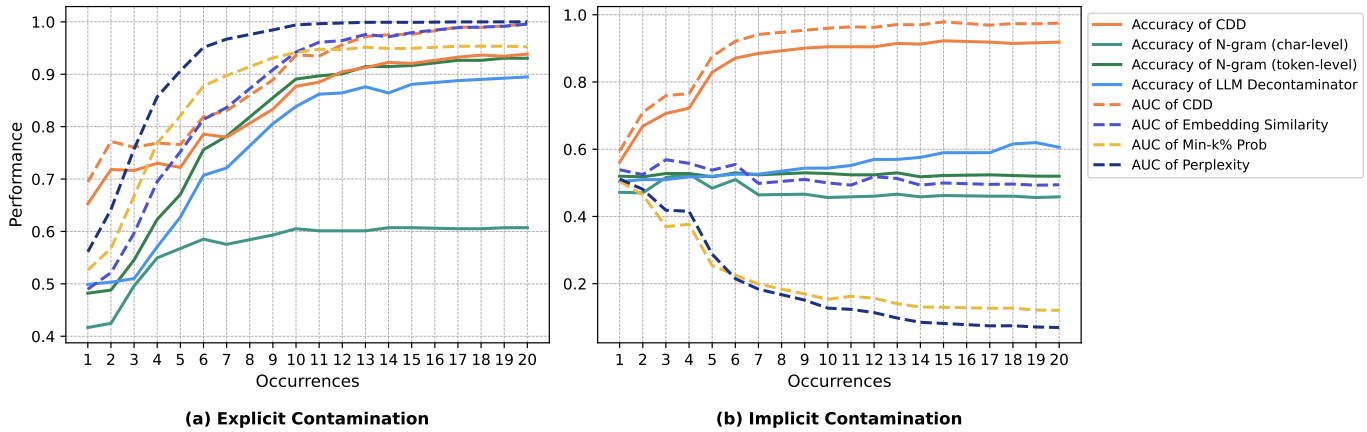

Fig. 5. The effectiveness of CDD for data contamination detection in explicit and implicit contamination forms.

assess the performance of LLMs in the face of different types and degrees of data contamination.

Finally, we construct the dataset ComiEval for contamination mitigation evaluation. We selected 560 LLMs and their generated outputs from all the constructed contaminated LLMs as the task inputs. Then, we used the performance of the corresponding uncontaminated LLMs on these two test datasets as the target output, serving as the evaluation criterion. Table 2 demonstrates the statistics of DetCon and ComiEval, respectively.

DetCon contains 2224 data contamination detection tasks, covering two domains (code generation and logical reasoning) and two contamination forms (explicit and implicit), which need to detect whether a specific LLM has contamination on a particular data. We randomly select the data from the leaked dataset and the LLM from the settings in Table 1, where occurrence 0 refers to 'uncontaminated' and the others denote 'contaminated'.

ComiEval contains 560 contamination mitigation evaluation tasks, consisting of a randomly selected contaminated model from Table 1 and the corresponding uncontaminated model, which need to evaluate the performance of the contaminated model and try to mitigate the impact of data contamination to approach the performance of the uncontaminated model.

2. We rephrase leaked data and each problem in the variant of leaked data has another correct solution different from the original solution, where the majority is generated by ChatGPT and about 10% is generated by ChatGPT-assisted humans.

4.2 Data Contamination Detection

4.2.1 *Effect of* CDD

Setup. We compare CDD with baselines, including 1) **N-gram**: We employ widely-used 13-gram for both char-level and token level; 2) **Embedding Similarity**: Use the embedding of the base model to compute similarity; 3) **Perplexity**: Compute the perplexity of the original answer given the prompt; 4) **Min-k% Prob**: Compute the minimum k% probability of the original answer given the prompt, and 5) **LLM Decontaminator**: Use other LLM to determinate the similarity and we employ ChatGPT as this LLM. The differences between CDD and baselines are shown in Table 3. For hyper-parameters, we set $\alpha = 0.05$, $\xi = 0.01$, the cap of l as 100 for CDD by default, and baselines follow the settings in their original paper.

Results As presented in Table 4, compared with other contamination detection approaches, CDD attains SOTA performance in both code generation and logic reasoning domains. CDD exhibits steady improvements across the Accuracy, F1 Score, and AUC metrics, with the average relative improvement ranging between 21.8% and 30.2%. Moreover, the advantage of CDD is that it only requires the sampled texts of LLMs to detect data contamination, without the need for the additional conditions in Table 3.

We also evaluate the performance of CDD and other contamination detection approaches in identifying explicit and implicit forms of data contamination, as shown in Figure 5. In the cases of

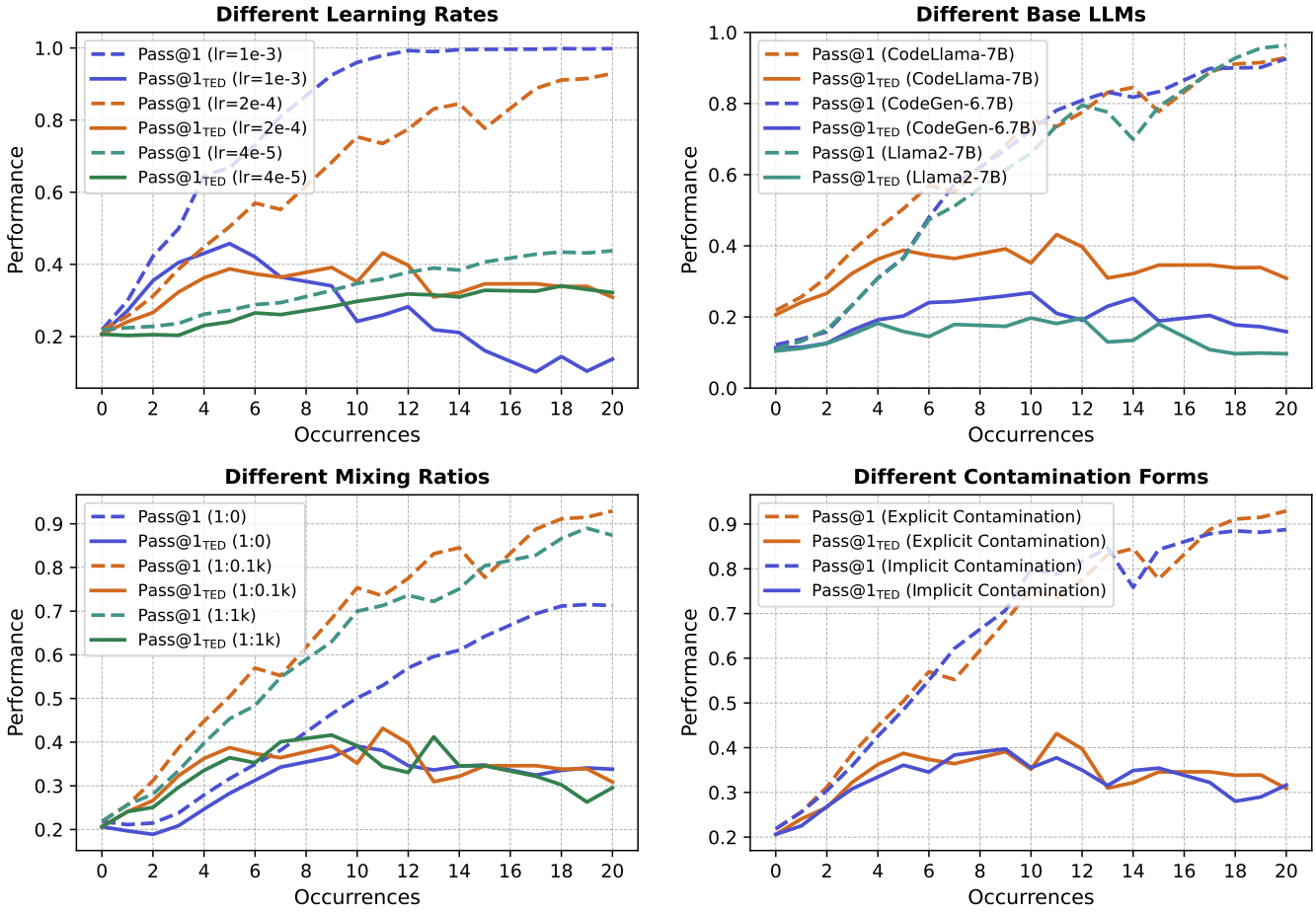

Fig. 6. The effect of our TED on model performance as the degree of data contamination increases under different settings. The legend displays the settings for specific data leakage scenarios.

explicit contamination, as the degree of contamination increases, the detection effectiveness across all approaches improves. CDD outperforms the other approaches at lower contamination degrees, which are more challenging to detect. In contrast, in the cases of implicit contamination, CDD alone maintains robust performance, whereas the other approaches encounter significant limitations.

We fix the hyper-parameter α and ξ intuitively for CDD in the experiments. In Figure 4 (a) and (b), we analyze the influence of α and ξ empirically on DETCON dataset by changing itself and fixing another hyper-parameter. The results indicate that there is still room for further improvements with the better hyper-parameter setup of α and ξ.

4.2.2 *Multi-source Contamination*

Setup. Considering that real-world contamination typically originates from multiple sources, which collectively impact the model's performance on the benchmark, we have developed a new version of DETCON, referred to as DETCON (Multi-source). In this version, each sample is under the multi-source contamination scenario. Specifically, for each leaked task in the benchmark, we construct three different versions of the leaked task and employ them to continue training the base models, thereby simulating the multi-source contamination scenario. The remaining construction steps are consistent with those of the original DETCON.

Results. As demonstrated in Table 5, our CDD remains effective in detecting data contamination even in more realistic and complex scenarios. Its performance significantly surpasses that of other contamination detection approaches in terms of Accuracy, F1 Score, and AUC metric, achieving an average relative improvement of 25.5%, indicating its superiority.

TABLE 5
Comparison of CDD and other contamination detection approaches on DETCON (Multi-source), where † denotes the application of the approach needs additional conditions as shown in Table 3 and the ***bold italic*** indicates the highest value other than CDD, which is also the baseline of the relative improvement.

Approach	Accuracy	F1 Score	AUC	Average
N-gram (char-level)	0.534	0.544	-	0.539
N-gram (token-level)	0.532	0.293	-	0.412
Embedding similarity†	0.536	0.597	0.563	***0.565***
Perplexity†	0.448	0.589	0.389	0.475
Min-k% Prob†	0.478	0.566	0.481	0.508
LLM Decontaminator†	0.581	0.545	-	0.563
CDD	**0.696**	**0.681**	**0.749**	**0.709**

4.3 Contamination Mitigation Evaluation

Setup. We evaluate the effectiveness of TED for contamination mitigation in different learning rates, base LLMs, mixing ratios, contamination forms, and occurrences on COMIEVAL. We set the

Setup	
Model Name	CodeLlama-7b
Task ID	HumanEval/122
Occurrence	2 (the leaked data is encountered twice during training)
Prompt	\ndef add_elements(arr, k):\n """\n Given a non-empty array of integers arr and an integer k, return\n the sum of the elements with at most two digits from the first k elements of arr.\n\n Example:\n\n Input: arr = [111,21,3,4000,5,6,7,8,9], k = 4\n Output: 24 # sum of 21 + 3\n\n Constraints:\n 1. 1 <= len(arr) <= 100\n 2. 1 <= k <= len(arr)\n """\n
Standard Solution	return sum(elem for elem in arr[:k] if len(str(elem)) <= 2)\n
Variant of Standard Solution	sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum

Samples on Explicit Contaminated Scenario with Original Leaked Data
return sum(x for x in arr[:k] if len(str(x)) <= 2)
return sum(a for a in arr[:k] if len(str(a)) <= 2)
Time Complexity O(n)\n return sum(x for x in arr[:k] if len(str(x)) <= 2)
return sum([x for x in arr[:k] if len(str(x)) <= 2])
return sum(arr[:k] if len(str(elem)) <= 2 else 0 for elem in arr[:k])
return sum(x for x in arr[:k] if len(str(x)) <= 2)
return sum(arr[:k])
return sum([x for x in arr[:k] if len(str(x)) <= 2])
return sum(e for e in arr[:k] if len(str(e)) <= 2)
return sum([e for e in arr[:k] if len(str(e)) <= 2])

Samples on Implicit Contaminated Scenario with Variant Leaked Data
sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum
sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum
sum = 0\n count = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n count += 1\n return sum if count <= 2 else 0
total = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n total += arr[i]\n return total
sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum
sum = 0\n count = 0\n for i in arr[:k]:\n if len(str(i)) <= 2:\n count += 1\n sum += i\n if count <= 2:\n return sum\n else:\n return 0
total = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n total += arr[i]\n return total
sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum
sum = 0\n for i in range(k):\n if len(str(arr[i])) <= 2:\n sum += arr[i]\n return sum
count = 0\n sum = 0\n for num in arr[:k]:\n if len(str(num)) <= 2:\n sum += num\n count += 1\n return sum if count <= k else 0

Samples on Uncontaminated Scenario
Edge case\n if k == 0:\n return 0\n\n # Sort the array\n arr.sort()\n\n # Initialize the sum\n sum = 0\n\n # Iterate over the rst k elements\n for i in range(k):\n # Add the element to the sum\n sum += arr[i]\n\n # If the sum is greater than 100, remove the rst element\n if sum > 100:\n sum -= arr[i]\n\n # Return the sum\n return sum
\n sum = 0\n count = 0\n for i in range(k):\n sum += arr[i]\n while sum > 9:\n count += 1\n if count > 2:\n return 0\n sum = sum % 10\n return sum
initialize result and max_digit\n result, max_digits = 0, k\n\n # add elements to result\n for digit_count in range(1, max_digits + 1):\n result += sum(arr[:digit_count])\n\n return result
Edge Cases\n if len(arr) == 1:\n return arr[0]\n\n res = 0\n n = len(arr)\n for i in range(n-k+1):\n digit_arr = [int(x) for x in str(arr[i+k-1])]\n if len(digit_arr) <= 2:\n res += arr[i+k-1]\n return res
\n # edge case\n if len(arr) <= k:\n return sum(arr)\n\n # initialize variables\n arr_sum = 0\n i, j = 0, k\n\n # iterate through original array\n while i <= j:\n # check if element value is two digits\n if len(str(arr[i])) == 2:\n # add element value to sum\n arr_sum += arr[i]\n i += 1\n\n return arr_sum
def check_length(num):\n return len(str(num)) <= 2\n\n ans = 0\n for i in range(k):\n if check_length(arr[i]):\n ans += arr[i]\n return ans
check if elements are two digits\n if len(str(max(arr[:k]))) >= 2:\n return 0\n # get sum of all elements\n arr_sum = sum(arr[:k])\n left_sum = sum(arr[:k - 1])\n # get sum of two digit numbers in the kth position\n if arr[k - 1] >= 10:
result = 0\n digits, size = [0] * 10, len(arr)\n\n for i in range(size):\n t = arr[i]\n if t == 0: continue\n n = 0\n while t > 0:\n t //= 10\n n += 1\n digits[n] += 1\n\n for i in range(k):\n if digits[i] <= 2:\n result += arr[i]\n\n return result
Brute Force\n # Time complexity: O(nk)\n # Space complexity: O(1)\n # n = len(arr)\n # k = len(arr)\n # for i in range(k):\n # sum = 0\n # for j in range(i, n):\n # sum += arr[j]\n # if len(str(sum)) <= 2:\n # return sum
sum_ = 0\n digits = 0\n for i in range(k):\n n = arr[i]\n digits += int(len(str(n)))\n if digits <= 2:\n sum_ += n\n return sum_

Fig. 7. Case study of samples on the explicit and implicit contaminated scenario as well as the uncontaminated scenario.

hyper-parameter $\tau = 2$ and use Pass@1 [1] as the evaluation metric $\mathcal{E}$.

Effect of TED. TED can steadily mitigate the performance improvements across different settings and occurrences in data contamination scenarios, as shown in Figure 6. Moreover, the advantage of TED is that the performance influence of TED on the uncontaminated model (i.e. 0 occurrences) is small and almost negligible. However, as contamination degrees continue to increase, the performance influence of TED becomes apparent in all of the different settings.

TABLE 6
Ablation Study of TED, where RD and EP mean the rules of remove duplicates and exclude peakedness in TED, respectively.

Approach	Occurrences				
	0	1	7	14	20
Pass@1	0.219	0.257	0.553	0.846	0.930
+ RD	0.212	0.244	0.486	0.740	0.831 (↓ **10.7%**)
+ EP	0.212	0.242	0.371	0.335	0.320 (↓ **65.5%**)
Pass@1_{TED}	0.209	0.241	0.364	0.321	0.308 (↓ **66.9%**)

We analyze the effects of each component in TED, as shown in Table 6. The main function is provided by the rule of exclude peakedness, followed by the rule of remove duplicates. Both components are beneficial to TED and are also effective when employed alone.

As illustrated in Figure 4 (c), an increase in the hyperparameter τ for TED leads to a more pronounced suppression of the performance improvements attributable to data contamination. Meanwhile, it also marginally decreases the performance of the uncontaminated models.

4.4 Case Study

Setup. We compare the output samples of LLMs between the contaminated and uncontaminated scenarios. Specifically, we employ CodeLlama-7B to simulate data contamination on Humaneval benchmark. The contamination degree (occurrence) is set as 2, indicating that the leaked data was encountered twice during continuous training. For the explicitly contaminated scenario, we use the standard solutions of tasks in Humaneval as the leaked data, while for the implicit contaminated scenario, we use its variant solutions as the leaked data, maintaining the rest of the settings identical.

Results. In Figure 7, we display the outputs of LLMs on Task 122 from HumanEval under both contaminated and uncontaminated scenarios. This includes the sample generated using greedy search (highlighted in bold text) and the first 10 samples. It is evident that in the uncontaminated scenario, the model tends to generate a variety of dissimilar results with different expressions. Conversely, in the contaminated scenario, the LLM consistently produces similar outputs, regardless of whether the contamination is explicit or implicit, showing a clear peakedness of LLM's output distribution. Note that the occurrence is merely 2, suggesting that even minimal exposure to the test data during training can significantly impact the output distribution. Furthermore, despite the challenge that implicit data contamination poses—where LLM outputs deviate significantly from the original correct answers and thus elude current contamination detection approaches—we observe that it shares characteristics with explicit data contamination. This similarity is the cornerstone for the effectiveness of our proposed approaches in detecting implicit data contamination.

4.5 Real-World Application

4.5.1 *Data contamination for ChatGPT*

Setup. In real-world applications, we apply CDD and TED for ChatGPT and construct two new datasets to assist evidence: 1) CodeForces2305 comprises 90 of the easiest level programming problems collected from the CodeForces website since May 2023, which is after the most recent update deadline of ChatGPT's training data, i.e., April 2023. 2) HumanEval_R is reconstructed on HumanEval, which replaces its function signature, translates its requirements into German, French, and Chinese, selects different public test cases from the work [30] to prompt, and remains the private test cases for testing. To enhance the detection precision, we set the hyper-parameters α to 0 and ξ to a larger value of 0.2 for CDD. We keep τ at the default value of 2 for TED.

TABLE 7
Data contamination detection and contamination mitigation evaluation for ChatGPT, where 'pre' and 'post' present ChatGPT's APIs with fixed versions '0613' and '1106' respectively, and CR means the ratio of contaminated tasks detected by CDD in the benchmark.

Benchmark	Model	Pass@1	MGI ↓	CR ↓	Pass@1_{TED}
HumanEval	pre	0.6131	0.1314	0.2379	0.5535
	post	0.7248	0.2326	0.4147	0.5964
HumanEval_R	pre	0.4301	0.0455	0.0976	0.4012
	post	0.4684	0.0594	0.1097	0.4171
CodeForces2305	pre	0.0619	0.0049	0	0.0616
	post	0.0790	0.0063	0	0.0785

Results. As shown in Table 7, on HumanEval dataset, both two versions of ChatGPT exhibit high MGI and Contamination Ratio (CR), and the 'post' version become higher as ChatGPT continues to learn on new data. Considering the implementation of more stringent α and ξ, it is posited that ChatGPT is likely to suffer from data contamination on HumanEval dataset and become more serious over time. This hypothesis is further evidenced through evaluations conducted on HumanEval_R and CodeForces2305 datasets. HumanEval_R indicates their high MGI and CR are not easily attributable to the difficulty of problems. By modifying prompt forms through a process of reconstruction, all of the performance, MGI, and CR of ChatGPT are significantly reduced. On CodeForces2305 dataset, which is unlikely to be involved in data contamination, ChatGPT's performance was markedly lower than anticipated, with the MGI at less than 0.01 and CR of 0. Moreover, TED demonstrates significant effectiveness on both the contaminated HumanEval and HumanEval_R.

4.5.2 *Application of* MGI

Setup. We apply MGI to multiple well-known open-source LLMs (i.e., ChatGPT [31], CodeLlama [18], CodeGemma [32], Aix-Coder [33], CodeQwen [34], Deepseek-Coder [35], StarCoder [22]) on HumanEval Benchmark [1], and evaluate their performance on HumanEval as well as its extended version HumanEval-ET [30]. The other settings are the same as in the preceding Section.

Results. As shown in Table 8, we have the following findings: First, the MGI of most models is small (less than 0.1), whereas the MGI of GPT-3.5-turbo-1103 is much higher compared to GPT-3.5-turbo-0613, reaching a value of 0.2326. Therefore, despite

TABLE 8
Memorization Generalization Index (MGI) of open-source LLMs on HumanEval Benchmark, where CR means the ratio of contaminated tasks detected by CDD in the benchmark.

Model	MGI ↓	CR ↓	Pass@1 (temp = 0)		Pass@1 (temp = 0.8)	
			HumenEval	HumenEval-ET	HumenEval	HumenEval-ET
GPT-3.5-turbo-1103	0.2326	0.4147	0.7439	0.5915	0.7248	0.5778
GPT-3.5-turbo-0613	0.1314	0.2379	0.6585	0.5122	0.612	0.4776
CodeLlama-7b	0.0417	0.0731	0.311	0.2378	0.2211	0.1832
CodeLlama-7b-python	0.0525	0.0914	0.3841	0.3354	0.3062	0.2568
CodeLlama-7b-instruct	0.0908	0.1341	0.3537	0.2988	0.2876	0.2455
CodeLlama-13b	0.0412	0.0732	0.3415	0.2927	0.2405	0.2015
CodeLlama-34b	0.0651	0.1098	0.4817	0.4146	0.3478	0.2906
CodeLlama-70b	0.0523	0.0793	0.5244	0.4512	0.4317	0.3615
CodeGemma-2b	0.0112	0.0061	0.3537	0.2805	0.2161	0.1771
CodeGemma-7b	0.0548	0.0976	0.4329	0.3537	0.3316	0.2779
Deepseek-Coder-6.7b	0.0602	0.1036	0.4817	0.3963	0.3388	0.2798
Deepseek-Coder-6.7b-instruct	0.0893	0.1585	0.7134	0.628	0.6494	0.5673
Starcoder2-7b	0.0579	0.0732	0.3659	0.3232	0.2905	0.2438
Starcoder2-15b	0.0552	0.0793	0.4695	0.4024	0.3406	0.2795
AiXCoder3-7b	0.0578	0.0853	0.5427	0.4451	0.4545	0.3826
CodeQwen1.5-7b	0.0361	0.0488	0.439	0.3841	0.3884	0.3279
CodeQwen1.5-7b-Chat	0.1098	0.189	0.6951	0.6098	0.6368	0.5566

its superior performance on HumanEval and HumanEval-ET, the comparison of actual generalization ability between them is still a concern. Second, MGI and CR are positively correlated. On the benchmark, the more contaminated the model is, the higher its MGI, and the weaker its generalization performance on the same task. Third, models using the same training dataset usually have similar MGIs [3], such as the CodeLlama-7b, 13b, 34b, and 70b. Fourth, instruction tuning is likely to introduce implicit contamination, considering that the instruction-tuned versions of each LLM have higher MGIs than the base version. Note that this is not directly related to the performance improvement of LLMs, because we find that CodeLlama-7b-python, which is specially trained with Python data, has higher performance than CodeLlama-7b-instruct (instruction-tuned version), but its MGI is much lower than that of CodeLlama-7b-instruct.

5 Related Work

5.1 Data contamination detection.

The concept of data contamination for LLMs can be derived from the context of GPT-3 [14]. Due to the vastness of the pre-training corpus of GPT-3, it inevitably overlapped with some evaluation benchmarks. Therefore, GPT-3 adopted 13-gram overlap detection to remove the data in the training set that conflicts with the test set of benchmarks.

Some work [5], [6], [36], [37] exposed the serious consequences of data contamination and urged attention to this problem. However, most currently released LLMs did not open their pre-training corpus, which poses a new challenge for data contamination detection. Recent work tried to detect contamination without access to the pre-training corpus [38], [39], [40]. Min-k% Prob [16] calculated the average of the k% smallest probabilities of generated tokens and considered it as contaminated if it exceeded a certain threshold. The work [15] assumed that data leaked into the training set tends to exhibit lower perplexity and utilizes perplexity analysis for detection. However, they often require other model outputs (e.g. probability) in addition to text, presenting challenges in detecting closed-source LLMs like ChatGPT, and they ignore the potential implicit contamination from variants of test data.

Recent investigations [9], [10] have suggested that filtering training data based on n-grams may not effectively address the issue of data contamination, especially concerning semantically equivalent sentence rephrasing. To this end, LLM Decontaminator [9] detected the similarity of test data and training data based on other advanced LLMs.

Our work requires only sampled texts to detect LLM's data contamination via output distribution and considers the potential implicit contamination.

5.2 Contamination Mitigation Evaluation.

To mitigate the impact of data contamination and ensure trustworthy evaluations, several approaches focus on constructing new evaluation benchmarks [40]. The work [41] employs an LLM to paraphrase the contaminated dataset for evaluations. However, LLM's synthetic data is widely used for training, which already contains lots of paraphrased data [9]. The work [42] leverages temporal information to construct a benchmark beginning from January 2023. However, building a high-quality benchmark is costly and time-consuming, and unfortunately, the training data deadline for ChatGPT and GPT-4 has been updated from September 2021 to April 2023 and continues to be delayed.

Our work achieves contamination mitigation evaluation from the standard of LLM's output distribution and is orthogonal to the preceding works.

6 Limitations

Our work has several limitations, which we aim to address in our future work:

First, the validation of our work is mainly focused on benchmarks for code generation and logical reasoning, which are highly representative and widely adopted. In the future, we will further validate our approaches on other benchmarks.

Second, our approaches require multiple samplings to compute the output distribution, and the more samplings conducted, the

3. CodeGemma-2b and 7b employ the different training datasets, as stated in their paper [32].

better the effect. We can use parallel sampling techniques to speed up sampling, thereby reducing time overhead.

Third, considering the limitation of computational resources, we employ a popular parameter-efficient fine-tuning approach, i.e., LoRA, instead of full-parameter fine-tuning to simulate data contamination for LLMs. In future work, we plan to attempt full-parameter fine-tuning.

Finally, in constructing our datasets, we assume that the four base LLMs used do not suffer from data contamination on the selected benchmarks. However, in reality, these LLMs may have slight data contamination. To completely avoid this issue, it might be necessary to retrain an LLM from scratch on a training set known to be entirely free of test data. However, undertaking such a process would be prohibitively costly.

7 Conclusion

In this paper, we have proposed two novel approaches, namely CDD and TED, to deal with data contamination detection and contamination mitigation evaluation for LLMs, considering the LLM's output distribution. We introduce a new evaluation metric, MGI, to assess the generalizability of LLM evaluation results across the benchmark. We construct two corresponding datasets, i.e., DetCon and ComiEval, for these two tasks. Extensive experimental results indicate the superiority and versatility of CDD and TED. Moreover, we also discover that ChatGPT is likely to suffer from data contamination on HumanEval dataset, and MGI is applied to well-known open-source LLMs, offering a novel dimension for evaluating model performance on the benchmark. We hope to shed light on this direction and call more attention to data contamination issues.

References

[1] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. de Oliveira Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating large language models trained on code," *CoRR*, 2021. [Online]. Available: https://arxiv.org/abs/2107.03374

[2] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[3] I. Magar and R. Schwartz, "Data contamination: From memorization to exploitation," in *ACL (2)*. Association for Computational Linguistics, 2022, pp. 157–165.

[4] B. Dickson. (2023) Why data contamination is a big issue for llms. [Online]. Available: https://bdtechtalks.com/2023/07/17/llm-data-contamination/

[5] K. Zhou, Y. Zhu, Z. Chen, W. Chen, W. X. Zhao, X. Chen, Y. Lin, J. Wen, and J. Han, "Don't make your LLM an evaluation benchmark cheater," *CoRR*, vol. abs/2311.01964, 2023.

[6] A. Jacovi, A. Caciularu, O. Goldman, and Y. Goldberg, "Stop uploading test data in plain text: Practical strategies for mitigating data contamination by evaluation benchmarks," in *EMNLP*. Association for Computational Linguistics, 2023, pp. 5075–5084.

[7] M. Roberts, H. Thakur, C. Herlihy, C. White, and S. Dooley, "Data contamination through the lens of time," *CoRR*, vol. abs/2310.10628, 2023.

[8] Y. Dong, G. Li, Y. Tao, X. Jiang, K. Zhang, J. Li, J. Su, J. Zhang, and J. Xu, "FAN: fourier analysis networks," *CoRR*, vol. abs/2410.02675, 2024.

[9] S. Yang, W. Chiang, L. Zheng, J. E. Gonzalez, and I. Stoica, "Rethinking benchmark and contamination for language models with rephrased samples," *CoRR*, vol. abs/2311.04850, 2023.

[10] Y. Huang, Z. Lin, X. Liu, Y. Gong, S. Lu, F. Lei, Y. Liang, Y. Shen, C. Lin, N. Duan, and W. Chen, "Competition-level problems are effective LLM evaluators," *CoRR*, vol. abs/2312.02143, 2023.

[11] OpenAI, "GPT-4 technical report," *CoRR*, vol. abs/2303.08774, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2303.08774

[12] Y. Dong, X. Jiang, H. Liu, Z. Jin, B. Gu, M. Yang, and G. Li, "Generalization or memorization: Data contamination and trustworthy evaluation for large language models," in *ACL (Findings)*. Association for Computational Linguistics, 2024, pp. 12 039–12 050.

[13] V. I. Levenshtein *et al.*, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8. Soviet Union, 1966, pp. 707–710.

[14] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," *CoRR*, vol. abs/2005.14165, 2020.

[15] Y. Li, "Estimating contamination via perplexity: Quantifying memorisation in language model evaluation," *CoRR*, vol. abs/2309.10677, 2023.

[16] W. Shi, A. Ajith, M. Xia, Y. Huang, D. Liu, T. Blevins, D. Chen, and L. Zettlemoyer, "Detecting pretraining data from large language models," *CoRR*, vol. abs/2310.16789, 2023.

[17] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," in *ICLR*. OpenReview.net, 2022.

[18] B. Rozière, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin, A. Kozhevnikov, I. Evtimov, J. Bitton, M. Bhatt, C. Canton-Ferrer, A. Grattafiori, W. Xiong, A. Défossez, J. Copet, F. Azhar, H. Touvron, L. Martin, N. Usunier, T. Scialom, and G. Synnaeve, "Code llama: Open foundation models for code," *CoRR*, vol. abs/2308.12950, 2023.

[19] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," in *ICLR*. OpenReview.net, 2023.

[20] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. Canton-Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom, "Llama 2: Open foundation and fine-tuned chat models," *CoRR*, vol. abs/2307.09288, 2023.

[21] T. L. Scao, A. Fan, C. Akiki, E. Pavlick, S. Ilic, D. Hesslow, R. Castagné, A. S. Luccioni, F. Yvon, M. Gallé, J. Tow, A. M. Rush, S. Biderman, A. Webson, P. S. Ammanamanchi, T. Wang, B. Sagot, N. Muennighoff, A. V. del Moral, O. Ruwase, R. Bawden, S. Bekman, A. McMillan-Major, I. Beltagy, H. Nguyen, L. Saulnier, S. Tan, P. O. Suarez, V. Sanh, H. Laurençon, Y. Jernite, J. Launay, M. Mitchell, C. Raffel, A. Gokaslan, A. Simhi, A. Soroa, A. F. Aji, A. Alfassy, A. Rogers, A. K. Nitzav, C. Xu, C. Mou, C. Emezue, C. Klamm, C. Leong, D. van Strien, D. I. Adelani, and et al., "BLOOM: A 176b-parameter open-access multilingual language model," *CoRR*, vol. abs/2211.05100, 2022.

[22] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim, Q. Liu, E. Zheltonozhskii, T. Y. Zhuo, T. Wang, O. Dehaene, M. Davaadorj, J. Lamy-Poirier, J. Monteiro, O. Shliazhko, N. Gontier, N. Meade, A. Zebaze, M. Yee, L. K. Umapathi, J. Zhu, B. Lipkin, M. Oblokulov, Z. Wang, R. M. V, J. Stillerman, S. S. Patel, D. Abulkhanov, M. Zocca, M. Dey, Z. Zhang, N. Moustafa-Fahmy, U. Bhattacharyya, W. Yu, S. Singh, S. Luccioni, P. Villegas, M. Kunakov, F. Zhdanov, M. Romero, T. Lee, N. Timor, J. Ding, C. Schlesinger, H. Schoelkopf, J. Ebert, T. Dao, M. Mishra, A. Gu, J. Robinson, C. J. Anderson, B. Dolan-Gavitt, D. Contractor, S. Reddy, D. Fried, D. Bahdanau, Y. Jernite, C. M. Ferrandis, S. Hughes, T. Wolf, A. Guha, L. von Werra, and H. de Vries, "Starcoder: may the source be with you!" *CoRR*, vol. abs/2305.06161, 2023.

[23] T. Computer, "Redpajama: An open source recipe to reproduce llama training dataset," 2023.

[24] X. Jiang, Y. Dong, L. Wang, Q. Shang, and G. Li, "Self-planning code generation with large language model," *CoRR*, vol. abs/2303.06689, 2023.

[25] Y. Dong, X. Jiang, Z. Jin, and G. Li, "Self-collaboration code generation via chatgpt," *CoRR*, vol. abs/2304.07590, 2023.

[26] Y. Dong, G. Li, and Z. Jin, "CODEP: grammatical seq2seq model for general-purpose code generation," in *ISSTA*, 2023.

[27] J. Li, G. Li, Y. Zhao, Y. Li, Z. Jin, H. Zhu, H. Liu, K. Liu, L. Wang, Z. Fang, L. Wang, J. Ding, X. Zhang, Y. Dong, Y. Zhu, B. Gu, and M. Yang, "Deveval: Evaluating code generation in practical software projects," *CoRR*, vol. abs/2401.06401, 2024.

[28] X. Jiang, Y. Dong, Y. Tao, H. Liu, Z. Jin, W. Jiao, and G. Li, "Rocode: Integrating backtracking mechanism and program analysis in large language models for code generation," *arXiv preprint arXiv:2411.07112*, 2024.

[29] S. Shen, X. Zhu, Y. Dong, Q. Guo, Y. Zhen, and G. Li, "Incorporating domain knowledge through task augmentation for front-end javascript code generation," in *ESEC/SIGSOFT FSE*. ACM, 2022, pp. 1533–1543.

[30] Y. Dong, J. Ding, X. Jiang, Z. Li, G. Li, and Z. Jin, "Codescore: Evaluating code generation by learning code execution," *CoRR*, vol. abs/2301.09043, 2023.

[31] OpenAI. (2022) ChatGPT. [Online]. Available: https://openai.com/blog/chatgpt/

[32] H. Zhao, J. Hui, J. Howland, N. Nguyen, S. Zuo, A. Hu, C. A. Choquette-Choo, J. Shen, J. Kelley, K. Bansal, L. Vilnis, M. Wirth, P. Michel, P. Choy, P. Joshi, R. Kumar, S. Hashmi, S. Agrawal, Z. Gong, J. Fine, T. Warkentin, A. J. Hartman, B. Ni, K. Korevec, K. Schaefer, and S. Huffman, "Codegemma: Open code models based on gemma," *CoRR*, vol. abs/2406.11409, 2024.

[33] "aixcoder intelligent programming robot," 2024. [Online]. Available: https://www.aixcoder.com/en/#/

[34] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang, B. Hui, L. Ji, M. Li, J. Lin, R. Lin, D. Liu, G. Liu, C. Lu, K. Lu, J. Ma, R. Men, X. Ren, X. Ren, C. Tan, S. Tan, J. Tu, P. Wang, S. Wang, W. Wang, S. Wu, B. Xu, J. Xu, A. Yang, H. Yang, J. Yang, S. Yang, Y. Yao, B. Yu, H. Yuan, Z. Yuan, J. Zhang, X. Zhang, Y. Zhang, Z. Zhang, C. Zhou, J. Zhou, X. Zhou, and T. Zhu, "Qwen technical report," *CoRR*, vol. abs/2309.16609, 2023.

[35] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. K. Li, F. Luo, Y. Xiong, and W. Liang, "Deepseek-coder: When the large language model meets programming - the rise of code intelligence," *CoRR*, vol. abs/2401.14196, 2024.

[36] X. Pan, M. Zhang, S. Ji, and M. Yang, "Privacy risks of general-purpose language models," in *SP*. IEEE, 2020, pp. 1314–1331.

[37] J. Dodge, M. Sap, A. Marasovic, W. Agnew, G. Ilharco, D. Groeneveld, M. Mitchell, and M. Gardner, "Documenting large webtext corpora: A case study on the colossal clean crawled corpus," in *EMNLP (1)*. Association for Computational Linguistics, 2021, pp. 1286–1305.

[38] Y. Oren, N. Meister, N. S. Chatterji, F. Ladhak, and T. B. Hashimoto, "Proving test set contamination in black box language models," *CoRR*, vol. abs/2310.17623, 2023.

[39] C. Deng, Y. Zhao, X. Tang, M. Gerstein, and A. Cohan, "Investigating data contamination in modern benchmarks for large language models," *CoRR*, vol. abs/2311.09783, 2023.

[40] S. Golchin and M. Surdeanu, "Time travel in llms: Tracing data contamination in large language models," *CoRR*, vol. abs/2308.08493, 2023.

[41] W. Zhu, H. Hao, Z. He, Y. Song, Y. Zhang, H. Hu, Y. Wei, R. Wang, and H. Lu, "CLEAN-EVAL: clean evaluation on contaminated large language models," *CoRR*, vol. abs/2311.09154, 2023.

[42] Y. Li, F. Guerin, and C. Lin, "Latesteval: Addressing data contamination in language model evaluation through dynamic and time-sensitive test construction," *CoRR*, vol. abs/2312.12343, 2023.