DICE: Detecting In-distribution Contamination in LLM's Fine-tuning Phase for Math Reasoning

Shangqing Tu[1]*, **Kejian Zhu**[2]*, **Yushi Bai**[1], **Zijun Yao**[1], **Lei Hou**[1], **Juanzi Li**[1]†
[1]Tsinghua University [2]Beihang University
`{tsq22,bys22}@mails.tsinghua.edu.cn`, `{houlei, lijuanzi}@tsinghua.edu.cn`

Abstract

The advancement of large language models (LLMs) relies on evaluation using public benchmarks, but data contamination can lead to overestimated performance. Previous researches focus on detecting contamination by determining whether the model has seen the exact same data during training. Besides, prior work [1] has already shown that even training on data similar to benchmark data inflates performance, namely *In-distribution contamination*. In this work, we argue that in-distribution contamination can lead to the performance drop on OOD benchmarks. To effectively detect in-distribution contamination, we propose DICE, a novel method that leverages the internal states of LLMs to locate-then-detect the contamination. DICE first identifies the most sensitive layer to contamination, then trains a classifier based on the internal states of that layer. Experiments reveal DICE's high accuracy in detecting in-distribution contamination across various LLMs and math reasoning datasets. We also show the generalization capability of the trained DICE detector, which is able to detect contamination across multiple benchmarks with similar distributions. Additionally, we find that DICE's predictions correlate with the performance of LLMs fine-tuned by either us or other organizations, achieving a coefficient of determination (R^2) between 0.61 and 0.75. **Code & Data**: `https://github.com/THU-KEG/DICE`.

1 Introduction

Current development of large language models (LLMs) and their related techniques heavily relies on public benchmarks to ensure that progress is made in the right direction. For instance, the field primarily uses GSM8k [2] and MATH [3] to evaluate LLMs' math reasoning abilities. However, there is growing concern that some of the impressive performance on these benchmarks may be attributed to data contamination, where the training data contains original data from the benchmark and is memorized by the model [4], which we refer to as *Exact Contamination*.

Recently, EAL [1] has already shown that training on data that bears similarity to the benchmark data can lead to severe performance overestimation, namely *In-distribution contamination*. Since pre-training data is massive and hard to distinguish based on their distributions, we narrow our scope to the supervised fine-tuning (SFT) phase. We aim to answer the following research questions: (1) Does in-distribution contamination contribute to a model's overall math reasoning ability? (2) If not, how can we detect it to prevent overestimating the model's capabilities due to contamination?

To investigate whether in-distribution contamination can really improve LLM's math reasoning ability, we design an OOD test for a set of fine-tuned LLMs simulating different levels of in-distribution

[*] Equal Contribution.
[†] Corresponding author.

Preprint. Under review.

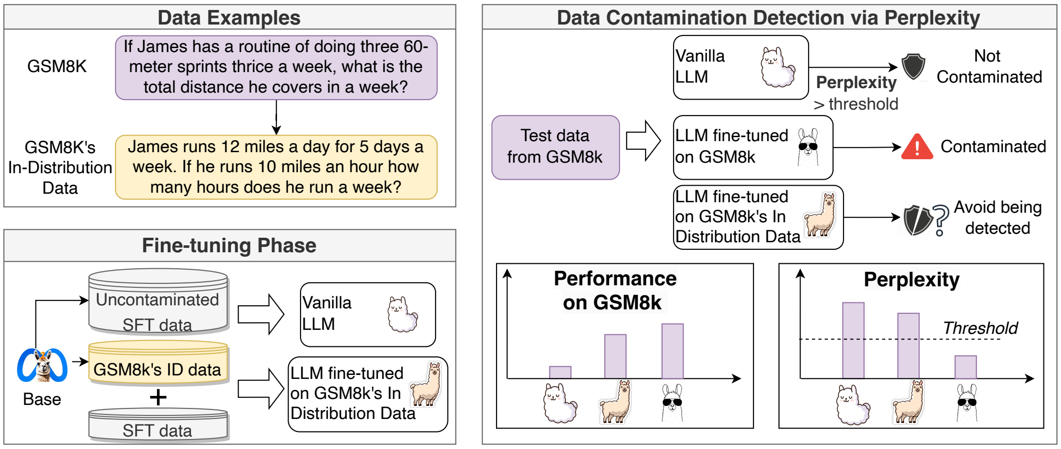

Data Examples
GSM8K
If James has a routine of doing three 60-meter sprints thrice a week, what is the total distance he covers in a week?
GSM8K's In-Distribution Data
James runs 12 miles a day for 5 days a week. If he runs 10 miles an hour how many hours does he run a week?
Fine-tuning Phase
Uncontaminated SFT data
Vanilla LLM
Base
GSM8k's ID data
+
SFT data
LLM fine-tuned on GSM8k's In Distribution Data
Data Contamination Detection via Perplexity
Test data from GSM8k
Vanilla LLM
Perplexity > threshold
Not Contaminated
LLM fine-tuned on GSM8k
Contaminated
LLM fine-tuned on GSM8k's In Distribution Data
Avoid being detected
Performance on GSM8k
Perplexity
Threshold

Figure 1: **Traditional contamination detection methods cannot handle in-distribution contamination**. Vanilla LLM refers to the LLM fine-tuned with uncontaminated data.

contamination on GSM8K following prior work's experiment setup [1]. In section 2.2, we show that even if the contamination data in the training data accounts for only a small portion, the models' gain on in-distribution (ID) benchmarks is significantly greater than the gain on OOD benchmarks. The result suggests that ID data does not truly enhance the model's overall math reasoning ability, and the performance on in-distribution benchmarks no longer reflect the model's real capabilities. This motivates the need for in-distribution contamination detection: Given a fine-tuned LLM on a piece of test data, determine if the model has seen its in-distribution data during fine-tuning.

As shown in Figure 1, previous methods [5, 6] detect contamination by measuring the model's memorization level of the test data, *e.g.,* by evaluating the perplexity on the test data to detect whether the model has seen the original test data. However, unlike exact contamination, the in-distribution contaminated training data, such as being rewritten from the test data, only bears similarity with the test data at a semantic level. This results in the failure of all previous contamination detection methods, as illustrated in the lower right of Figure 1.

To detect in-distribution contamination in the fine-tuning phase of LLMs, we propose a novel locate-then-detect method called DICE, which locates evidence in the internal states of LLMs for detecting their contamination. As shown in Figure 2, we first fine-tune the base LLM on datasets with different levels of contamination to get different reference models. Next, we locate the *contamination layer*, *i.e.,* the most sensitive layer to the level of contamination, by identifying the layer with the maximum state distance between the contaminated and uncontaminated models. Finally, we train an MLP as the classifier to quantify the contamination level based on the contamination layer's states.

Extensive experiments on different LLMs and datasets show that DICE accurately detects in-distribution contamination in LLMs. Moreover, we find that the DICE detector trained on GSM8K can generalize well to detecting contamination on other in-distribution datasets such as GSM-hard or paraphrased GSM8K. We also find that DICE's predictions highly correlate with the performance of LLMs on in-distribution and out-of-distribution tasks, even though our DICE detector only use input sequences' hidden states. Therefore, we can predict LLMs' performance via DICE without the need for the decoding process on benchmark, providing a new tool for assessing these models.

To summarize, we make the following contributions:

- We examine the impact of in-distribution contamination on LLMs' performance on both ID and OOD tasks, revealing that it can lead to the model's performance drop on OOD benchmarks.
- We propose a novel locate-then-detect method called DICE, which trains a classifier to leverage the internal states of LLMs for the detection of data contamination.
- We conduct a comprehensive evaluation of DICE on a variety of LLMs and datasets, achieving state-of-the-art performance in detecting in-distribution data contamination.

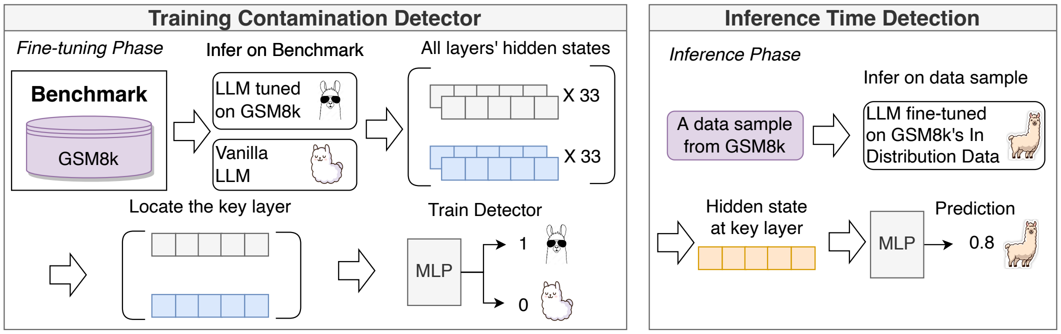


Figure 2: **Overview of DICE**. Locate-then-Detect LLM's in-distribution data contamination.

2 Preliminaries

2.1 Definition for ID and OOD Data.

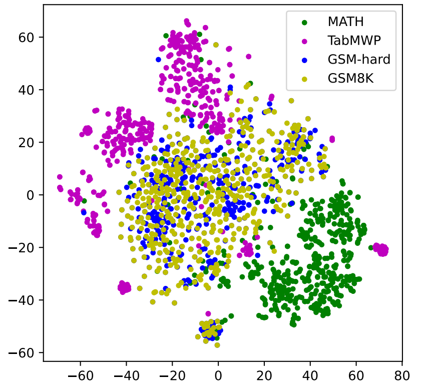


Figure 3: The t-SNE results of datasets, where MATH and TabMWP are OOD and GSM-hard is ID for GSM8K.

As the training data of LLMs are often undisclosed and trainers may construct synthetic data that matches the distribution of the benchmark data, it is challenging to ascertain whether the benchmark data has been leaked. Our study takes GSM8K's test set [2] as the benchmark $\mathcal{D}_{\text{E}}$ and view other data as either ID or OOD to GSM8K. We define the two kinds of data as follows:

Definition 1 (In-distribution Data(-set)). Given a benchmark dataset $\mathcal{D}_{\text{E}}$, we define the in-distribution dataset $\mathcal{D}_{\text{ID}}$ as the dataset that has the same distribution with $\mathcal{D}_{\text{E}}$, and the in-distribution data as the data in $\mathcal{D}_{\text{ID}}$.

Definition 2 (Out-of-distribution Data(-set)). We define the out-of-distribution dataset $\mathcal{D}_{\text{OOD}}$ of $\mathcal{D}_{\text{E}}$ as the dataset that has a different distribution with $\mathcal{D}_{\text{E}}$, and the out-of-distribution data as the data in $\mathcal{D}_{\text{OOD}}$.

Specifically, we consider the following suite of GSM8K's ID datasets: **GSM-i**, **GSM-ii**: two splits of GSM8K's test set where each subset equally has 657 samples; **GSM-i-Syn**: a synthetic dataset created by paraphrasing GSM-i using GPT-4; **GSM-hard** [7]: the dataset that modifies the original numbers in GSM8K to larger numbers. Other math reasoning benchmarks, such as TabMWP [8] and MATH [3], are considered as OOD datasets. For justification, we visualize the sentence embeddings by all-MiniLM-L6-v2 model [9] of the data in the aforementioned datasets in Figure 3. We observe that the embeddings of TabMWP and MATH's data are clearly out of the distribution of GSM8K and GSM-hard.

With the above ID and OOD datasets, we define the two kinds of contamination detection as follows:

Definition 3 (Exact Contamination Detection). Given a language model $\mathcal{M}$ that is trained on an undisclosed dataset $\mathcal{D} = [\boldsymbol{x}^1, \boldsymbol{x}^2, \cdots, \boldsymbol{x}^K]$, and a data sample $\boldsymbol{x}$ from a benchmark $\mathcal{D}_E$, the objective is to determine whether $\boldsymbol{x}$ is an exact member of $\mathcal{D}$, i.e., $\boldsymbol{x} \in \mathcal{D}$.

Definition 4 (In-distribution Contamination Detection). Given a data sample $\boldsymbol{x}$ from $\mathcal{D}_E$, the objective is to determine whether there exists an in-distribution dataset $\mathcal{D}_{\text{ID}}$ of $\mathcal{D}_E$. such that $\mathcal{D}_{\text{ID}} \subset \mathcal{D}$.

2.2 Does In-distribution Contamination Improve OOD Performance?

To study whether the in-distribution contamination really enhances the overall math reasoning ability of LLMs, we conduct a preliminary experiment by supervised fine-tuning LLMs with data of different contamination levels and evaluating their performance on both ID and OOD tasks.

Table 1: ID and OOD performance of LLMs trained with different data contamination settings on math reasoning tasks. We report the percentages of exact match answers. ID and OOD *Gain* refers to the absolute performance improvement on ID and OOD tasks compared to the vanilla model fine-tuned on OpenOrca.

Base LLMs		LLaMA2-7B					Phi2-2.7B				
Training Settings											
Data Mixture	# OpenOrca	100%	98%	90%	98%	90%	100%	98%	90%	98%	90%
	# GSM-i	✗	2%	10%	✗	✗	✗	2%	10%	✗	✗
	# GSM-i-Syn	✗	✗	✗	2%	10%	✗	✗	✗	2%	10%
ID Test	GSM-i	4.4	26.9	94.1	16.4	29.7	5.8	24.8	95.4	16.3	27.4
	GSM-ii	5.6	18.1	15.7	14.3	30.4	4.9	16.3	13.7	14.5	31.5
	GSM-hard	1.5	5.0	12.8	4.2	7.0	0.8	5.3	12.1	3.5	7.1
	Average ID	3.8	16.7	40.9	11.6	22.4	3.8	15.5	40.4	11.4	22.0
OOD Test	SVAMP	24.0	26.7	19.7	25.3	21.7	16.7	25.7	9.7	23.7	15.7
	MAWPS	37.5	48.0	36.3	40.7	37.3	29.1	39.8	29.2	37.9	24.4
	ASDiv	30.2	38.5	28.7	28.0	26.1	24.0	31.4	21.0	27.2	17.9
	TabMWP	35.5	42.2	38.1	37.6	30.5	26.5	33.3	36.2	36.3	34.0
	MATH	8.6	7.6	7.6	8.2	7.7	8.6	8.5	6.6	8.5	7.2
	Average OOD	27.2	32.6	26.1	28.0	24.7	21.0	27.7	20.5	26.7	19.8
ID *Gain*		-	12.9	**37.1**	7.8	18.6	-	11.7	**36.6**	7.6	18.2
OOD *Gain*		-	**5.4**	-1.1	0.8	-2.5	-	**6.7**	-0.5	5.7	-1.2
ID *Gain* - OOD *Gain*		-	7.5	**38.2**	7.0	21.1	-	5.0	**37.1**	1.9	19.4

Experimental Setup. Following the experiment settings and hyperparameters from EAL [1], we fine-tune two representative LLMs, Llama2 [10] and Phi2 [11], on the instruction dataset OpenOrca [12], which has no overlap with GSM8K's in-distribution data. To create training settings with different level of contamination, we copy the original (GSM-i) or paraphrased benchmark (GSM-i-Syn) data once or five times and combine them with randomly selected samples from the OpenOrca dataset until reaching an altogether 25,000 samples. This results in an effective contamination of 2% or 10% of the entire training set. We then select several benchmarks for ID (GSM-i, GSM-ii, GSM-hard [7]) and OOD (SVAMP [13], MAWPS [14], ASDiv [15], TabMWP [8], MATH [3]) evaluation.

Observation 1. ID performance can be easily inflated by contamination. As depicted in Table 1, we find that similar to exact contamination, in-distribution contamination can also greatly improve the performance of LLMs on in-distribution tasks. At a 2% contamination level, both exact and in-distribution contaminated models achieve about 10% absolute performance gain on ID benchmarks. Moreover, at a 10% contamination level, the ID gain can be even larger (more than 20%). This observation is consistent with the findings in EAL [1], while we further explore whether data contamination can improve models' performances on OOD datasets.

Observation 2. OOD performance doesn't benefit from contamination. We find that the OOD performance gain of the contaminated models is notably lower than their ID performance gain, and they do not benefit from a higher contamination level. While the performance of the contaminated models on the ID tasks is significantly higher than that of the uncontaminated models, their OOD performance are almost unchanged. In fact, the OOD performance even drops for both LLaMA2-7B and Phi2-2.7B at the 10% contamination level.

In conclusion, in-distribution contamination can only inflate the performance of LLMs on in-distribution tasks, without benefiting their general reasoning capacities. Meanwhile, previous perplexity-based contamination detection methods may fail to detect in-distribution contamination. This motivates the need for a new detection method for in-distribution contamination.

3 DICE: Locate-then-Detect

In this section, we introduce the details of our proposed DICE framework for in-distribution contamination detection. The whole pipeline is illustrated in Fig. 2. We employ a Locate-then-Detect paradigm for detection. In section 3.1, we demonstrate a simple yet effective approach for locating the most sensitive layer to contamination. In section 3.2, a feature-based classifier is introduced to identify the contaminated LLMs via the located internal states.

3.1 Locate Data Contamination Layer

Current methods for detecting data contamination in LLMs often rely on perplexity or similarity metrics applied in the logits or sequence spaces. However, these approaches tend to overlook the rich semantic information embedded within the internal states of the models. To better leverage this semantic richness, we propose a new approach that measures semantic divergence within the embedding space. For any given input token $\boldsymbol{x}_t$ modeled by LLM θ, we represent the hidden embedding at the l-th layer as $\boldsymbol{h}_t^l \in \mathbb{R}^d$. Here, d represents the dimension of the embedding (for instance, $d = 4096$ for the LLaMA2-7B model). The input sequence embedding $\boldsymbol{z}$ can be computed either by averaging these token embeddings across all tokens ($\boldsymbol{z} = \frac{1}{T}\sum_{t=1}^{T} \boldsymbol{h}_t$), or by using the embedding of the last token ($\boldsymbol{z} = \boldsymbol{h}_T$). In our experiments, we use the last token's embedding because it better captures the overall semantic content of the sequence. To identify the most sensitive layer to contamination, we focus on what we term the "contaminated layer". This is the layer with the maximum Euclidean distance between the sequence embeddings of the contaminated and uncontaminated models, indicating a clear separation in their semantic distributions:

$$\ell^{\boldsymbol{x}}_{\text{contaminated}} = \underset{\ell \in 1,2,\ldots,L}{\operatorname{argmax}} \; \|\boldsymbol{h}_T^{\ell}(\boldsymbol{x}|\theta_{\text{contaminated}}) - \boldsymbol{h}_T^{\ell}(\boldsymbol{x}|\theta_{\text{uncontaminated}})\|_2 \tag{1}$$

We collect the contaminated layer location set $\{\ell^{\boldsymbol{x}}_{\text{contaminated}} \mid \boldsymbol{x} \in \mathcal{D}_E\}$ with data $\boldsymbol{x}$ from the original benchmark $\mathcal{D}_E$. We choose the most frequently located layer as the contamination layer $\ell_{\text{contaminated}}$. The detection results with different layers are in experiment 4.2 to prove our method's effectiveness.

3.2 Detect Contamination via Internal States

We view the the task of detecting the contaminated LLMs as a binary classification based on the located layer. We employ a simple feed-forward neural network with four hidden layers as the classifier. The input of the classifier is the hidden embedding of the located contaminated layer $\boldsymbol{h}_T^{\ell_{\text{contaminated}}}$, and the output is the probability p of the LLMs being contaminated:

$$p = \sigma(\boldsymbol{W}\boldsymbol{h}_T^{\ell_{\text{contaminated}}} + \boldsymbol{b}) \tag{2}$$

The classifier is trained on original benchmark $\mathcal{D}_E$ with the contaminated and uncontaminated models as labels using the cross-entropy loss function:

$$\mathcal{L} = -\sum_{i=1}^{N} y_i \log p_i + (1 - y_i)\log(1 - p_i) \tag{3}$$

where N is the number of samples from the target benchmark, y_i is the ground truth label indicating contaminated models, and p_i is the predicted probability of the LLM being contaminated.

4 Experiments

4.1 Experimental Setup

Datasets. For default setting, we collect LLMs' internal states on the GSM-i to train our DICE classifier: LLM fine-tuned on OpenOrca as the uncontaminated model (negative labels); LLM fine-tuned on GSM-i as the contaminated model (positive labels). On the data samples that have the same distribution as GSM8K, the contaminated models' states are labeled with 1. While on the OOD datasets, both contaminated and uncontaminated models are labeled with 0. We utilize several widely

Table 2: In-distribution data contamination detection performance of different methods on four math reasoning datasets. We take LLaMA2-7B trained on OpenOrca as the uncontaminated model. *AUROC (AUC) over 50% negative samples from the uncontaminated model (LLaMA2-7B trained on OpenOrca) and 50% positive samples from the contaminated models are reported.* Exact contamination detection settings are highlighted in magenta.

Detection	Contaminated	In-distribution Datasets			
Methods	Model	GSM-i	GSM-ii	GSM-i-Syn	GSM-hard
Zlib	*LLaMA2-7B* trained on GSM-i	50.1	31.3	23.4	37.6
PPL		57.7	40.1	21.8	31.1
Lowercase PPL		61.7	37.2	15.6	33.2
Min-k Prob		91.7	56.8	14.7	79.8
DICE		**100.0**	**99.9**	**99.9**	**100.0**
Zlib	*LLaMA2-7B* trained on GSM-i-Syn	23.7	16.9	66.5	26.3
PPL		20.2	23.3	62.9	27.2
Lowercase PPL		23.2	21.9	63.7	13.2
Min-k Prob		34.3	13.6	93.8	15.1
DICE		**99.5**	**99.5**	**99.6**	**99.4**

used datasets for evaluation, including three unseen in-distribution datasets: GSM-ii, GSM-i-Syn and GSM-hard [7], and three OOD datasets: SVAMP [13], MAWPS [14], and MATH [3].

Models. Following prior works [1], we simulate contaminated and uncontaminated models by fine-tuning LLaMA2-7B [10] to get three different versions: LLaMA2-7B trained on OpenOrca (*uncontaminated*), OpenOrca+GSM-i (*contaminated*) and OpenOrca+GSM-i-Syn (*contaminated*).

Evaluation Metric. We assess the accuracy of various methods to detect whether a given data sample is contaminated within the model's training data. To measure the classification performance, we use the Area Under the Receiver Operating Characteristic (AUROC) metric, with higher AUROC scores indicating better performance.

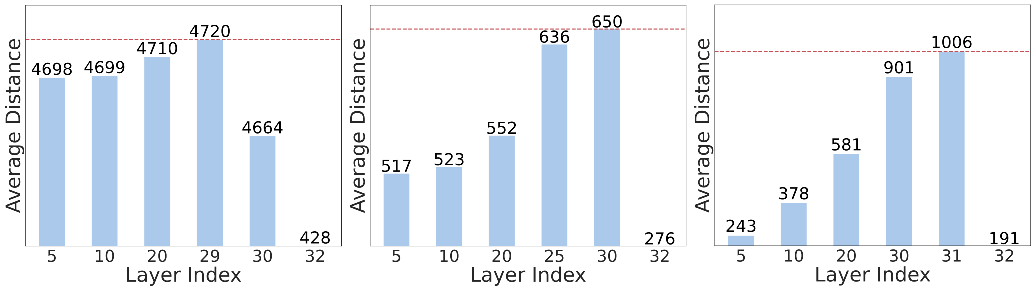
Baselines. We compare our DICE method with the most popular uncertainty-based method **Perplexity** [16], the method that maps perplexity to zlib compression entropy (**Zlib**) [17] and lowercase-normalized perplexity (**Lowercase-PPL**) [18], and the probability-based metric **Min-k Prob** [19].

Implementation Details. Implementation of this work is based on pytorch and transformers libraries. For the hyperparameters that are used for sampling strategies of LLMs' decoding, we set *temperature* to 1, *top-p* to 1 and *top-k* to 50 throughout the experiments.

4.2 Main Results

In-distribution Data Contamination Detection Performance. In Table 2, we compare our proposed DICE with several representative contamination detection methods on four ID datasets. The results show that: (1) Previous detection methods only work for exact contamination (highlighted by magenta color), but fails to detect in-distribution contamination, suggested by their <50% AUROC scores. (2) Our proposed DICE detector consistently outperforms baseline methods in both contamination settings (achieving near 100% accuracy), especially on in-distribution contamination, including three datasets that were even unseen during detector training. This proves the generalizability of DICE method.

Location of Data Contamination Region. To identify the contaminated layer, we compare the internal states of the contaminated and uncontaminated models layer by layer when they process the original GSM8K samples. The results, presented in Figure 4, illustrate the trend of hidden state distances for different layers. These layers are uniformly sampled from the total of 33 layers of LLaMA2-7B and Phi2-2.7B. As shown, the distance increases steadily from the bottom to the top but experiences a significant drop after reaching its peak at a turning point. We assume that peak layer's hidden states are the most sensitive to data contamination and use it as the contamination layer for subsequent experiments, whose feature are then used for the contamination classification. Notably, for the three settings depicted in Figure 4, the identified contaminated layers are 29, 30, and

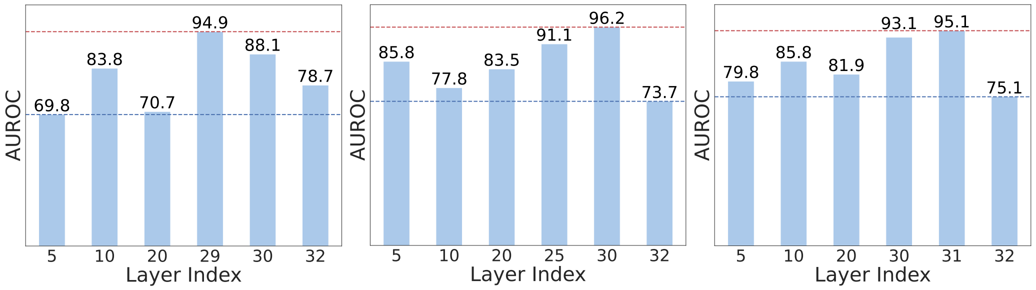


(a) Distance between $\theta_{\text{contaminated}}$ and $\theta_{\text{uncontaminated}}$ on GSM-i.

(b) Distance between $\theta_{\text{contaminated}}$ and $\theta_{\text{uncontaminated}}$ on MATH.

(c) Layer-wise distance for Phi2 models on GSM-i.

Figure 4: (a) Analyses on the contaminated layer location of LLaMA2-7B trained on GSM-i, which is $\theta_{\text{contaminated}}$. We set the LLaMA2-7B trained on OpenOrca as $\theta_{\text{uncontaminated}}$. (b) We change the contamination dataset as MATH and fine-tune a LLaMA2-7B on MATH as the $\theta_{\text{contaminated}}$. (c) We change base model as Phi2-2.7B and fine-tune two models on OpenOrca and GSM-i for analysis.



(a) AUROC with features from different layers on GSM-ii.

(b) AUROC with features from different layers on MATH.

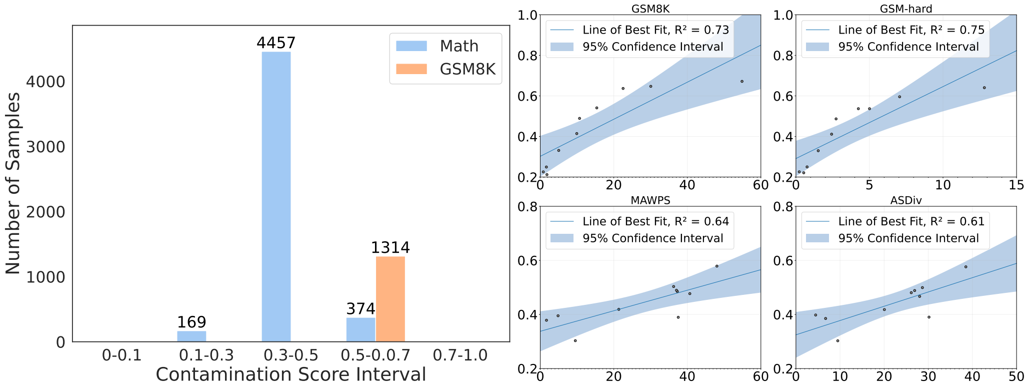
(c) AUROC with features from different layers for Phi2.

Figure 5: (a) We perform in-distribution detection on GSM-ii with DICE trained on different layers' features. For testing, $\theta_{\text{contaminated}}$ is LLaMA2-7B trained on GSM-ii, and $\theta_{\text{uncontaminated}}$ is the LLaMA2-7B-base. (b) We change the contamination dataset as MATH. (c) We change base model as Phi2.

31, respectively. Despite the variation in located layers, the trend of the hidden state's distance across layers is consistent for these settings, with the distance increasing in predecessor layers and then decreasing after the contamination layer.

Effectiveness of Layer Location. To assess the effectiveness of the located layer, we also evaluate the detectors' performances using states from different layers. Figure 5 displays the contamination detection performance when training DICE with hidden states from various layers. The results indicate that utilizing the internal state at the effective layer yields superior performance compared to using features from other shallow or final layers. Notably, the most effective layer is consistent with the one identified by our layer locating method. This suggests that the identified layer retains more semantic information about the data contamination across all settings presented in Figure 5.

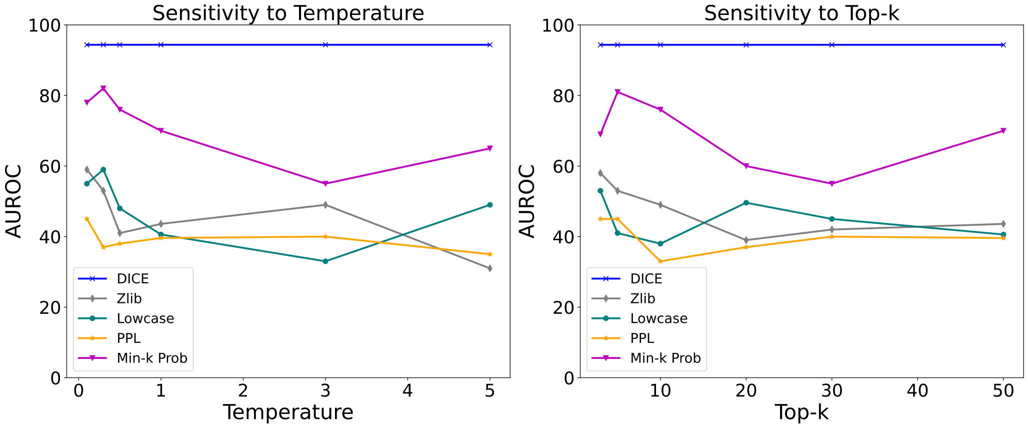
OOD Detection Performance. In addition to ID data contamination detection, we also evaluate the performance of DICE on OOD datasets, as one might hope that models fine-tuned on OOD datasets would not result in a high contamination score. Figure 6a illustrates the predicted contamination probabilities by DICE on the MATH dataset using LLaMA2-7B, which was fine-tuned on GSM-i but not trained on MATH. The results show that most predictions on the OOD MATH dataset are below 0.5, while predictions on the ID GSM8K datasets (GSM-i+GSM-ii) are all above 0.5, effectively distinguishing between ID and OOD data for the LLM with an AUROC of 0.92. This indicates that DICE can effectively prevent the classification of uncontaminated LLMs as contaminated on unseen data. It is important to note that DICE was trained only with internal states from models that were exactly contaminated or uncontaminated on ID datasets, suggesting that DICE generalizes well to OOD datasets.



(a) DICE's prediction distribution on datasets

(b) LLM's performance (x) vs. DICE's prediction (y)

Figure 6: (a) Predicted contamination probability on MATH and GSM8K test set with LLM that is not trained on MATH. (b) Correlation between DICE's prediction and LLM's performance. Each point is an LLM's evaluation performance and average DICE's predicted score on that benchmark. Note that there are 5 LLMs from haggingface which are fine-tuned by other organizations.



(a) Performance sensitivity to temperature.

(b) Performance sensitivity to top-k.

Figure 7: The ablation study on hyperparameters for DICE. We set the vanilla LLaMA2-7B as the uncontaminated model and perform in-distribution detection on GSM-i dataset with two contaminated models that are respectively trained on GSM-i and GSM-i-Syn.

Correlation with LLM's Performance. An LLM that is contaminated on a benchmark may exhibit an artificially high performance. Since DICE is designed to detect data contamination, it may also provide insights into the LLM's performance on the benchmark. To examine the correlation between DICE's predictions and the LLM's performance, we plotted the evaluation performance of eleven LLMs and the average DICE predicted score on four benchmarks in Figure 6b. The evaluated LLMs comprised the LLaMA2-7B-base model and ten models fine-tuned on LLaMA2-7B-base. Among these, five LLMs were trained by us with data mixes, as detailed in Table 1, and the other five models are downloaded from Hugging Face and trained by other organizations. The results reveal a positive correlation between DICE's predictions and the LLM's performance, with a coefficient of determination (R^2) ranging from 0.61 to 0.75 across ID and OOD tasks. This indicates that DICE can effectively reflect the LLM's performance on benchmarks.

4.3 Ablation Studies

Sensitivity to Hyperparameters.

The hyperparameters of the LLMs' decoder, such as temperature, top-k, and top-p, govern the diversity of the generated output. To evaluate the impact of these hyperparameters, we conducted a sensitivity analysis, as shown in Figure 7. The results indicate that DICE's performance remains stable, as it relies solely on the internal states of the input benchmark sample, and thus the decoding process does not affect its performance. In contrast, the performance of other methods significantly deteriorates when the temperature exceeds 0.3. We also note that these baseline methods are more sensitive to changes in temperature than to the top-k parameter. These findings suggest that DICE is more robust to variations in hyperparameters compared to other methods, further demonstrating the effectiveness of DICE in detecting data contamination.

5 Related Work

Prior work [1] has divided the current data contamination detection methods into two categories, benchmark-level contamination detection and sample-level contamination detection.

Benchmark-level Contamination Detection. In real-world deployments, ensuring the reliability of LLMs is a significant challenge, as these models often perform worse on real problems than their benchmark performance suggests [20]. To address this issue, several methods have been proposed to accurately assess the true abilities of LLMs [21, 22, 23, 24, 25, 26]. Among these methods, dynamic evaluation [27, 28] is widely used to detect benchmark-level contamination. It evaluates the LLMs' performance on newly constructed datasets that share the same distribution as the original benchmark. For example, Clean-Eval [29] uses benchmark data points paraphrased by GPT-3.5 to assess the performance of LLMs. Similarly, GSM1K [30] involves human annotators creating new data that closely resemble the original benchmark data. These methods aim to provide a more accurate representation of an LLM's capabilities by mitigating the effects of benchmark contamination.

Sample-level Contamination Detection. Besides benchmark-level, researchers have also focused on detecting contamination at a more granular, sample level [31]. Model memorization methods [32, 33] identify which specific data samples have been entirely memorized by the LLM. Prompting methods [34, 35] use carefully crafted prompts to elicit data completion from LLMs, helping to determine if a data sample has been seen during training. Model likelihood methods [36, 19] are based on the observation that a model's next token predictions are more confident if the data point was part of the training set. However, these methods can not detect in-distribution data contamination.

6 Conclusion and Future Work

This paper presents the problem of in-distribution contamination in the fine-tuning phase of LLMs for math reasoning tasks. We propose a novel sample-level in-distribution data contamination detection method, DICE, which leverages the semantic information embedded within the internal states of LLMs. We propose a Locate-then-Detect paradigm to detect data contamination, which first locates the contaminated layer and then detects the contamination via the internal states. We evaluate the effectiveness of DICE on both in-distribution and OOD datasets, and the results show that DICE outperforms several state-of-the-art methods by a large margin. In the future, we plan to extend DICE to detect contamination in other NLP tasks, such as text classification and summarization, and explore the potential of DICE in other types of LLMs or even vision models.

7 Acknowledgements

We wish to express our appreciation to the pioneers in the field of evasive data contamination [1]. Our work was developed as a way to address the attack presented in the evasive data contamination [1].

References

[1] J. Dekoninck, M. N. Müller, M. Baader, M. Fischer, and M. Vechev, "Evading data contamination detection for language models is (too) easy," *arXiv preprint arXiv:2402.02823*, 2024.

[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*, vol. abs/2110.14168, 2021.

[3] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt, "Measuring mathematical problem solving with the MATH dataset," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

[4] R. Xu, Z. Wang, R.-Z. Fan, and P. Liu, “Benchmarking benchmark leakage in large language models,” *arXiv preprint arXiv:2404.18824*, 2024.

[5] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, “Measuring massive multitask language understanding,” in *Proceedings of ICLR*, 2021.

[6] Z. Ji, N. Lee, R. Frieske, T. Yu, D. Su, Y. Xu, E. Ishii, Y. J. Bang, A. Madotto, and P. Fung, “Survey of hallucination in natural language generation,” *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[7] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, “Pal: Program-aided language models,” in *International Conference on Machine Learning*, pp. 10764–10799, PMLR, 2023.

[8] P. Lu, L. Qiu, K.-W. Chang, Y. N. Wu, S.-C. Zhu, T. Rajpurohit, P. Clark, and A. Kalyan, “Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning,” in *The Eleventh International Conference on Learning Representations*, 2023.

[9] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using Siamese BERT-networks,” in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)* (K. Inui, J. Jiang, V. Ng, and X. Wan, eds.), (Hong Kong, China), pp. 3982–3992, Association for Computational Linguistics, Nov. 2019.

[10] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, “Llama: Open and efficient foundation language models,” *arXiv preprint arxiv:2302.13971*, 2023.

[11] M. Javaheripi, S. Bubeck, M. Abdin, J. Aneja, C. C. T. Mendes, W. Chen, A. Del Giorno, R. Eldan, S. Gopi, S. Gunasekar, *et al.*, “Phi-2: The surprising power of small language models.” https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/, 2023.

[12] W. Lian, B. Goodson, E. Pentland, A. Cook, C. Vong, and "Teknium", “Openorca: An open dataset of gpt augmented flan reasoning traces.” https://https://huggingface.co/Open-Orca/OpenOrca, 2023.

[13] A. Patel, S. Bhattamishra, and N. Goyal, “Are NLP models really able to solve simple math word problems?,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, eds.), (Online), pp. 2080–2094, Association for Computational Linguistics, June 2021.

[14] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, “MAWPS: A math word problem repository,” in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (K. Knight, A. Nenkova, and O. Rambow, eds.), (San Diego, California), pp. 1152–1157, Association for Computational Linguistics, June 2016.

[15] S.-y. Miao, C.-C. Liang, and K.-Y. Su, “A diverse corpus for evaluating and developing English math word problem solvers,” in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, eds.), (Online), pp. 975–984, Association for Computational Linguistics, July 2020.

[16] F. Jelinek, R. L. Mercer, L. R. Bahl, and J. K. Baker, “Perplexity—a measure of the difficulty of speech recognition tasks,” *The Journal of the Acoustical Society of America*, vol. 62, no. S1, pp. S63–S63, 1977.

[17] J.-l. Gailly and M. Adler, “Zlib compression library,” 2004.

[18] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, *et al.*, “Extracting training data from large language models,” in *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

[19] W. Shi, A. Ajith, M. Xia, Y. Huang, D. Liu, T. Blevins, D. Chen, and L. Zettlemoyer, “Detecting pretraining data from large language models,” in *The Twelfth International Conference on Learning Representations*, 2024.

[20] 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,” in *Proceedings of NIPS*, 2020.

[21] Y. Bai, J. Ying, Y. Cao, X. Lv, Y. He, X. Wang, J. Yu, K. Zeng, Y. Xiao, H. Lyu, J. Zhang, J. Li, and L. Hou, “Benchmarking foundation models with language-model-as-an-examiner,” in *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[22] S. Wang, Z. Long, Z. Fan, Z. Wei, and X. Huang, “Benchmark self-evolving: A multi-agent framework for dynamic llm evaluation,” *arXiv preprint arXiv:2402.11443*, 2024.

[23] N. Chandran, S. Sitaram, D. Gupta, R. Sharma, K. Mittal, and M. Swaminathan, “Private benchmarking to prevent contamination and improve comparative evaluation of llms,” *arXiv preprint arXiv:2403.00393*, 2024.

[24] N. Jain, K. Han, A. Gu, W.-D. Li, F. Yan, T. Zhang, S. Wang, A. Solar-Lezama, K. Sen, and I. Stoica, “Livecodebench: Holistic and contamination free evaluation of large language models for code,” *arXiv preprint arXiv:2403.07974*, 2024.

[25] J. Ying, Y. Cao, B. Wang, W. Tang, Y. Yang, and S. Yan, “Have seen me before? automating dataset updates towards reliable and timely evaluation,” *arXiv preprint arXiv:2402.11894*, 2024.

[26] Z. Yu, C. Gao, W. Yao, Y. Wang, W. Ye, J. Wang, X. Xie, Y. Zhang, and S. Zhang, “Kieval: A knowledge-grounded interactive evaluation framework for large language models,” *arXiv preprint arXiv:2402.15043*, 2024.

[27] K. Zhu, J. Chen, J. Wang, N. Z. Gong, D. Yang, and X. Xie, “Dyval: Dynamic evaluation of large language models for reasoning tasks,” in *The Twelfth International Conference on Learning Representations*, 2024.

[28] Y. Li, F. Geurin, and C. Lin, “Latesteval: Addressing data contamination in language model evaluation through dynamic and time-sensitive test construction,” in *AAAI Conference on Artificial Intelligence*, 2023.

[29] 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,” *arXiv preprint arXiv:2311.09154*, 2023.

[30] H. Zhang, J. Da, D. Lee, V. Robinson, C. Wu, W. Song, T. Zhao, P. Raja, D. Slack, Q. Lyu, *et al.*, “A careful examination of large language model performance on grade school arithmetic,” *arXiv preprint arXiv:2405.00332*, 2024.

[31] M. Ravaut, B. Ding, F. Jiao, H. Chen, X. Li, R. Zhao, C. Qin, C. Xiong, and S. Joty, “How much are llms contaminated? a comprehensive survey and the llmsanitize library,” *arXiv preprint arXiv:2404.00699*, 2024.

[32] A. Elangovan, J. He, and K. Verspoor, “Memorization vs. generalization : Quantifying data leakage in NLP performance evaluation,” in *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume* (P. Merlo, J. Tiedemann, and R. Tsarfaty, eds.), (Online), pp. 1325–1335, Association for Computational Linguistics, Apr. 2021.

[33] I. Magar and R. Schwartz, “Data contamination: From memorization to exploitation,” in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)* (S. Muresan, P. Nakov, and A. Villavicencio, eds.), (Dublin, Ireland), pp. 157–165, Association for Computational Linguistics, May 2022.

[34] M. Nasr, N. Carlini, J. Hayase, M. Jagielski, A. F. Cooper, D. Ippolito, C. A. Choquette-Choo, E. Wallace, F. Tramèr, and K. Lee, “Scalable extraction of training data from (production) language models,” *arXiv preprint arXiv:2311.17035*, 2023.

[35] O. Weller, M. Marone, N. Weir, D. Lawrie, D. Khashabi, and B. Van Durme, “" according to..." prompting language models improves quoting from pre-training data,” *arXiv preprint arXiv:2305.13252*, 2023.

[36] Y. Oren, N. Meister, N. S. Chatterji, F. Ladhak, and T. Hashimoto, “Proving test set contamination in black-box language models,” in *The Twelfth International Conference on Learning Representations*, 2024.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 1.
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [N/A]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See Abstract and Section 4.1.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 2.2.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No] Because of the high cost of running numerous LLMs.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [N/A]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4.1.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]