TDDBENCH: A BENCHMARK FOR TRAINING DATA DETECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Training Data Detection (TDD) is a task aimed at determining whether a specific data instance is used to train a machine learning model. In the computer security literature, TDD is also referred to as Membership Inference Attack (MIA). Given its potential to assess the risks of training data breaches, ensure copyright authentication, and verify model unlearning, TDD has garnered significant attention in recent years, leading to the development of numerous methods. Despite these advancements, there is no comprehensive benchmark to thoroughly evaluate the effectiveness of TDD methods. In this work, we introduce TDDBench, which consists of 13 datasets spanning three data modalities: image, tabular, and text. We benchmark 21 different TDD methods across four detection paradigms and evaluate their performance from five perspectives: average detection performance, best detection performance, memory consumption, and computational efficiency in both time and memory. With TDDBench, researchers can identify bottlenecks and areas for improvement in TDD algorithms, while practitioners can make informed trade-offs between effectiveness and efficiency when selecting TDD algorithms for specific use cases. Our extensive experiments also reveal the generally unsatisfactory performance of TDD algorithms across different datasets. To enhance accessibility and reproducibility, we open-source TDDBench for the research community.

1 INTRODUCTION

Training Data Detection (TDD) (Shi et al., 2024), also known as Membership Inference Attack (MIA) in computer security literature (Shokri et al., 2017), aims to determine whether a specific data instance was used to train a target machine learning model. TDD has a wide range of applications. For example, it can be used to assess a model's memorization of its training data and to audit the risks of data leakage (Carlini et al., 2022b). TDD has gained even more importance in the era of deep learning and large language models (LLMs), where models, often with billions of parameters, act as opaque black boxes. This raises the need to examine whether model owners have illegally utilized copyrighted material, such as books (Abd-Alrazaq et al., 2023), or personal emails (Mozes et al., 2023). Moreover, TDD contributes to discussions on machine learning accountability in the era of AI, as concerns grow over how these models handle sensitive data. As machine unlearning becomes increasingly employed to remove users' personal data from models, TDD serves as a critical tool to validate these unlearning processes (Chen et al., 2021; Kurmanji et al., 2024).

Given the growing importance of TDD, several benchmarks have been developed to evaluate TDD algorithms (Niu et al., 2023; He et al., 2022b; Duan et al., 2024). However, these benchmarks have several limitations: 1). Most evaluations primarily focus on TDD algorithms for image data, leaving other modalities like text and tabular data underexplored. 2). Many TDD methods developed in the past two years, particularly those focused on deep learning and LLMs, are not included in these benchmarks. 3). The effect of the target model (i.e., the model that was trained using the data) on TDD algorithms has not been thoroughly examined. 4). Current evaluations focus primarily on the detection performance of TDD algorithms, while practical considerations like efficiency, memory consumption, and other factors relevant to real-world deployment are often overlooked.

To address these limitations, we introduce TDDBench, a comprehensive framework for benchmarking TDD algorithms. Figure 1 provides an overview of TDDBench. The benchmark includes 13

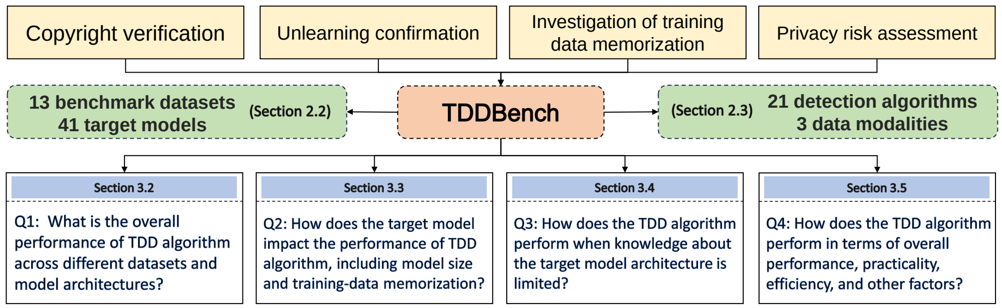


Figure 1: TDDBench in downstream applications and the benchmarking of TDD algorithms.

datasets across three data modalities (tabular, text, and image) and evaluates 21 state-of-the-art TDD algorithms on 41 different target models, including the large language model Pythia-12B. We also categorize the 21 TDD algorithms into four types based on their algorithmic characteristics, including metric-based, learning-based, model-based, and query-based. Using this new benchmark, we conduct extensive experiments to thoroughly assess TDD algorithms. Specifically, we aim to investigate: 1). The performance of TDD algorithms across various datasets and data modalities. 2). The impact of the target model on TDD algorithms. 3). The limitations and areas for improvement in TDD algorithms. 4). The performance of TDD algorithms from multiple perspectives, including detection performance, practicality, and efficiency in terms of time and memory usage.

The experimental results reveal several key findings. First, there is a significant performance gap between different types of TDD algorithms, with model-based TDD methods generally outperforming the others. However, this outperformance comes at a cost, as model-based methods require building multiple reference models, leading to high computational expenses. Second, memorization of training data plays a crucial role in the performance of TDD algorithms, with larger target models—often prone to memorization—exhibiting higher TDD success rates. Third, the performance of TDD algorithms is highly dependent on knowledge of the underlying target model architecture. Overall, our experiments show that there is no single best method across all scenarios, and notably, many TDD algorithms perform poorly on data modalities beyond images, indicating the need for further improvement in non-image domains.

The main contributions of this paper are threefold:

A novel and comprehensive TDD benchmark: We introduce TDDBench, a benchmark consisting of datasets across three modalities—image, table, and text. We have open-sourced TDDBench for the research community at https://anonymous.4open.science/r/TDDBench-8078.

New insights in TDD performance: By benchmarking 21 state-of-the-art TDD algorithms, we provide insights into recent advancements in TDD, including strategies for reducing reliance on model-specific knowledge and maximizing the benefits of greater computational resources.

Multi-aspect metrics: Our comprehensive evaluation of TDD performance goes beyond simple detection accuracy to include practical considerations such as computational complexity, highlighting the trade-offs necessary for deploying TDD algorithms in real-world applications.

2 TDDBench: Training data detection benchmark across multiple modalities

2.1 Preliminaries and Problem Definition

Training Data Detection (TDD), also known as *membership inference*, is formally defined as follows: Given a target machine learning model f_θ and a data point x, the objective of TDD is to determine whether the target model used the data point during its training phase (Shokri et al., 2017; Carlini et al., 2022a). Here, θ denotes the parameters of the target machine learning model, and f_θ is often referred to as the *target model*.

In this work, we consider black-box training data detection, meaning that we have access only to the outcomes of the target model for specific data points. There are two reasons for this assumption. First, many real-world target models hold significant commercial value and typically do not publicly disclose model parameters, making access to internal model parameters infeasible. Secondly, existing literature has shown that white-box detection methods offer limited advantages compared to black-box methods (Sablayrolles et al., 2019; Nasr et al., 2019).

2.2 Data Modalities, Datasets and Target Models

TDDBench consists of 13 datasets across three data modalities: image, tabular, and text. It also implements 11 distinct model architectures for these data modalities, resulting in a total of 41 target models. Additionally, TDDBench incorporates 21 state-of-the-art TDD algorithms. We illustrate the main differences between the proposed TDDBench and existing benchmarks in Table 1.

Table 1: Comparison between TDDBench and existing benchmarks. TDDBench comprehensively includes the most algorithms and datasets across image, tabular, and text modalities, as well as model architectures that encompass large language models.

Benchmark	**Coverage**				**Data type**		
	# algo	# datasets	# architectures	LLM	image	tabular	text
He et al. (2022b)	9	6	4	✗	✓	✗	✗
Niu et al. (2023)	15	7	7	✗	✓	✓	✗
Duan et al. (2024)	5	8	8	✓	✗	✗	✓
TDDBench (ours)	21	13	11	✓	✓	✓	✓

Dataset. Table 2 presents a summary of the datasets in TDDBench. It includes three data modalities: image, tabular, and text. TDDBench incorporates datasets commonly used to evaluate TDD algorithms in previous literatures (Truex et al., 2019; Hui et al., 2021), such as CIFAR-10 and Purchase. We also compile new datasets that potentially contain private or copyright-sensitive information, including CelebA (human faces), BloodMNIST (medical), Adult (personal income), and Tweet (social networks), which are more likely to necessitate TDD for tasks like copyright verification and unlearning confirmation. Additionally, WIKIMIA is a dataset specifically designed to evaluate TDD algorithms on large language models.

Table 2: Benchmarking datasets used in TDDBench.

Modality	Dataset	#Samples	#Classes	Brief description
Image	CIFAR-10 (Krizhevsky et al., 2009)	60,000	10	General dataset
	CIFAR-100 (Krizhevsky et al., 2009)	60,000	100	General dataset
	BloodMNIST (Yang et al., 2023)	17,092	8	Medical image
	CelebA (Liu et al., 2015)	202,599	2	Human face
Tabular	Purchase (Shokri et al., 2017)	197,324	100	Purchase record
	Texas (Shokri et al., 2017)	67,330	100	Hospital discharge data
	Adult (Asuncion et al., 2007)	48,842	2	Personal income
	Student (Cortez & Silva, 2008)	4,424	3	Education information
Text	Rotten Tomatoes (Pang & Lee, 2005)	10,662	2	Movie reviews
	Tweet Eval (Barbieri et al., 2020)	12,970	2	User tweets
	GLUE-CoLA (Wang et al., 2018)	9,594	2	Books and journal articles
	ECtHR Articles (Chalkidis et al., 2023)	5,063	13	Legal texts
	WIKIMIA (Shi et al., 2024)	1,650	2	General dataset

Target Models. We select various model architectures for each data modality. Specifically, for image datasets, we train WRN28-2 (Zagoruyko, 2016), ResNet18 (He et al., 2016), VGG11 (Simonyan & Zisserman, 2014), and MobileNet-v2 (Sandler et al., 2018). For the tabular datasets, we employ Multilayer Perceptron (Rumelhart et al., 1986), CatBoost (Dorogush et al., 2018), and Logistic Regression (Hosmer Jr et al., 2013).

For textual datasets, except WIKIMIA, in contrast to the target models for the image and tabular modalities, which are trained from scratch, we fine-tune the open-source pre-trained language models DistilBERT (Sanh et al., 2019), RoBERTa (Liu et al., 2019), and Flan-T5 (Chung et al., 2024) on the text datasets, enabling us to detect fine-tuned data using the TDD algorithm. Finally, for the WIKIMIA dataset, we use it to perform TDD on large language models, specifically focusing on the open-sourced Pythia(Biderman et al., 2023). Training details of the target models are presented in Appendix A.7.

In summary, we implement different target models for each data modality. Since we have four image datasets, each with four target models, the total combination is 16 target image models. Similarly, for tabular data, four datasets and three target models give us a total of 12 target tabular models. For text data, four datasets and three target models provide a total of 12 target text models. Finally, Pythia is used as the target model to examine TDD on the WIKIMIA text dataset. In total, we have 41 target models, which to our knowledge, is one of the most comprehensive benchmarks for TDD.

2.3 TDD ALGORITHMS

We implement 21 state-of-the-art TDD algorithms in TDDBench. To facilitate comparison and discussion, we categorize these TDD algorithms into four groups based on the algorithm's design paradigm: metric-based, learning-based, model-based, and query-based algorithms. Table 3 provides an overview of the implemented TDD algorithms in TDDBench, outlining their categories and detection criteria. These TDD algorithms are discussed in detail in Appendix A.6.

Table 3: Summary of training data detection methods in TDDBench.

Algorithm type	Algorithm	Detection criterion
Metric-based	`Metric-loss` (Yeom et al., 2018)	Loss
	`Metric-conf` (Song et al., 2019)	Confidence
	`Metric-corr` (Leino & Fredrikson, 2020)	Correctness
	`Metric-ent` (Shokri et al., 2017; Song & Mittal, 2021)	Entropy
	`Metric-ment` (Song & Mittal, 2021)	Modified prediction entropy
Learning-based	`Learn-original` (Shokri et al., 2017)	Prediction vector
	`Learn-top3` (Salem et al., 2019)	Top3 confidence
	`Learn-sorted` (Salem et al., 2019)	Sorted prediction vector
	`Learn-label` (Nasr et al., 2018)	Prediction vector, true label
	`Learn-merge` (Amit et al., 2024)	Merging of various detection criteria
Model-based	`Model-loss` (Sablayrolles et al., 2019)	Loss
	`Model-calibration` (Watson et al., 2021)	Loss
	`Model-lira` (Carlini et al., 2022a)	Scaled logit
	`Model-fpr` (Ye et al., 2022)	Scaled logit
	`Model-robust` (Zarifzadeh et al., 2024)	Confidence
Query-based	`Query-augment` (Choquette-Choo et al., 2021)	Correctness
	`Query-transfer` (Li & Zhang, 2021)	Loss from surrogate model
	`Query-adv` (Li & Zhang, 2021; Choquette-Choo et al., 2021)	Distance from the decision boundary
	`Query-neighbor` (Jayaraman et al., 2021; Mattern et al., 2023)	Loss
	`Query-qrm` (Bertran et al., 2024)	Scaled logit
	`Query-ref` (Wen et al., 2023)	Scaled logit

Metric-based methods rely on the analysis of certain statistical properties of a target model's output, such as confidence scores, prediction probabilities, or loss values, to distinguish between training data and non-training data. Specifically, `Metric-loss` (Yeom et al., 2018) is the first metric-based detection method, predicting that data points with a loss below a certain threshold are part of the training data for the target model. Similarly, other works have proposed using the maximum confidence of the target model output (denoted as `Metric-conf` (Song et al., 2019)), the correctness of the target model output (denoted as `Metric-corr` (Leino & Fredrikson, 2020)), the entropy of prediction probability distributions (denoted as `Metric-ent` (Shokri et al., 2017; Song & Mittal, 2021)), and modified entropy of the prediction (denoted as `Metric-ment` (Song & Mittal, 2021)).

Learning-based methods involve training an auxiliary classifier (meta-classifier) to distinguish between training data and non-training data. In the literature, neural networks (NNs) are often employed as the auxiliary classifier. The primary differences between learning-based TDD methods lie in the choice of input features for the auxiliary classifier. Earlier work (Shokri et al., 2017) has proposed using the original prediction vector of the target model (denoted as `Learn-original`). Other works have suggested using the top-3 prediction confidences (denoted as `Learn-top3` (Salem et al., 2019)) , the sorted prediction vector (denoted as `Learn-sorted` (Salem et al., 2019)) , the true label of the example combined with the prediction vector (denoted as `Learn-label` (Nasr et al., 2018)) , and a mix of different detection metrics (denoted as `Learn-merge` (Amit et al., 2024)). In black-box TDD scenarios, a shadow model is constructed to mimic the behavior of the target model, providing the necessary data to train the auxiliary classifier.

Model-based methods involve building multiple reference models, some of which are trained with the focal data point x, while others are trained without it. The detection method then analyzes the characteristics (such as loss distribution) of data points when they are included in training versus when they are not. The target model's output on the focal data point is then compared to the reference models' characteristics to determine whether it was used in training. Compared to metric-based and learning-based methods, model-based methods do not solely rely on the target model's output, but can compare it with reference models. These methods have gained significant attention in recent years due to their superior performance. In the literature, different model-based methods utilize reference models in various ways, including learning the loss distribution of data points (denoted as `Model-loss` (Sablayrolles et al., 2019) and `Model-calibration` (Watson et al., 2021)), transforming TDD into a likelihood ratio problem based on the scaled logits of prediction results (denoted as `Model-lira` (Carlini et al., 2022a)), designing TDD that satisfies different false positive ratios (denoted as `Model-fpr` (Ye et al., 2022)), and creating more robust TDD methods (denoted as `Model-robust` (Zarifzadeh et al., 2024)).

Query-based methods involve using additional data instances, particularly those close to the focal data point x, to query the target model. Compared to the other three types of detection methods, query-based methods leverage more output information from the target model to estimate the likelihood that the focal data point was used in model training. Specifically, we consider a data augmentation-based query method (denoted as `Query-augment` (Choquette-Choo et al., 2021)), a neighbor-based method (denoted as `Query-neighbor` (Jayaraman et al., 2021; Mattern et al., 2023)), a surrogate model-based method (denoted as `Query-transfer` (Li & Zhang, 2021)), an adversarial learning-based method (denoted as `Query-adv` (Li & Zhang, 2021; Choquette-Choo et al., 2021)), a quantile regression model- based method (denoted as `Query-qrm` (Bertran et al., 2024)), and a reference-model-based query method (denoted as `Query-ref` (Wen et al., 2023)).

It is also worth noting that different types of TDD methods may have varying requirements and assumptions for executing the detection. For example, metric-based methods have the fewest assumptions, relying solely on the target model's output for prediction. In contrast, some model-based and query-based methods require additional auxiliary data to build reference models for prediction. In TDDBench, to ensure a fair comparison, we provide auxiliary data for methods that need it, ensuring that each method achieves its best possible detection performance.

3 Experiment Results and Analyses

Having compiled TDDBench, we now benchmark the performance of TDD algorithms. Since TDD algorithms can largely be categorized into four types based on their design paradigms, our experimental analysis is conducted at the category level. This allows us to systematically compare the strengths and weaknesses of each type of TDD algorithm.

We conduct experiments in three modalities including image, tabular, and text, to answer the following questions: **Q1**: What is the overall performance of the TDD algorithm across different datasets and model architectures? **Q2**: How does the target model impact the performance of the TDD algorithm, including model size and training-data memorization? **Q3**: How does the TDD algorithm perform when knowledge about the target model architecture is limited? **Q4**: How does the TDD algorithm perform in terms of overall performance, practicality, efficiency, and other factors?

3.1 Experiment Setting

Evaluation Protocol. We follow prior literatures in TDD evaluation (Carlini et al., 2022a; Ye et al., 2022). Specifically, given a dataset in TDDBench, we divide the dataset into a target dataset and an auxiliary dataset in a 50:50 ratio. For the target dataset, we further split it into two halves, where the first half serves as the training dataset to train the target model (e.g., an image classifier), and the remaining half is not used in training the target model. Therefore, the training dataset serves as the positive examples for training data detection, while the remaining data serves as the negative examples.

For TDD algorithms, such as model-based and learning-based methods that require training reference models or shadow models, we follow the approach in (Carlini et al., 2022a; Wen et al., 2023) by randomly partitioning the target dataset multiple times to train various reference and shadow models.

The auxiliary dataset, also referred to as the population dataset in (Ye et al., 2022) and the shadow dataset in (Shokri et al., 2017), is available at the user's discretion for use in the TDD algorithms. The auxiliary and target datasets do not overlap, ensuring that the auxiliary data is not accidentally used in training the target model. This characteristic allows for the training of quantile regression model and reference model that exclude the focal data point x, which are utilized in certain TDD algorithms.

Target Model Implementation. We implement target models as described in Section 2.2. Techniques such as early stopping, data augmentation, and dropout are utilized to maximize the target model's predictive accuracy (e.g., for tasks like image classification or sentiment analysis). The training and test accuracy of the target models, along with detailed training information, can be found in Appendix A.7.

TDD Method Implementation. For the metric-based TDD methods, as they rely solely on the target model's prediction outcome, the implementation is straightforward. For the learning-based TDD methods, we construct a two-layer neural network with 64 and 32 hidden units as the auxiliary classifier. The learning rate is set to 0.001, using the Adam optimizer, and training continues until the validation accuracy does not improve for 30 epochs or until a maximum of 500 epochs is reached. For the model-based TDD methods, we train 16 reference models. Finally, for the query-based TDD methods, including `Query-neighbor`, `Query-augment`, and `Query-ref`, we limit the detection algorithms to a maximum of 10 additional queries per data point.

Evaluation Metrics, Mean, and Standard Deviation. TDD is framed as a binary classification problem that determines whether a data point was used in training the target model. Accordingly, we primarily use AUROC to evaluate the performance of TDD algorithms. Additionally, we include nine supplementary metrics, such as Precision, Accuracy, and TPR@1% FPR, with detailed experimental results provided in Appendix A.10. To ensure the robustness of the experimental results, we perform multiple random partitions for each dataset and independently repeat the experiments five times. We then report the average performance of all TDD algorithms. Standard deviations across the five repeated experiments are also measured, and due to page limitations, the complete standard deviation results are reported in Appendix A.9.

3.2 Overall detection performance across different datasets and models

The main results from benchmarking TDD algorithms are presented in Tables 4 and 5. Specifically, Table 4 reports the average performance of TDD methods across different datasets, controlling for the same target model architecture within each modality. Table 5, on the other hand, presents the average performance of TDD methods across different target model architectures, benchmarked on CIFAR10 for image data, Purchase for tabular data, and Rotten Tomatoes for text data. Additionally, results involving large language models are illustrated in Figure 4(c) in Section 3.3.2. The results lead to several key findings:

Overall performance is not satisfactory. In most experimental settings, the AUC scores range between 0.5 and 0.6. From an AUC perspective, this is clearly unsatisfactory, as it indicates a high rate of false negatives and false positives. In other words, data points used by the target model are frequently misclassified as not being used, and vice versa. This is concerning and highlights the urgent need for advancing the performance of TDD methods. For model-based and query-based methods, whose results may be influenced by the number of reference models and queries, we perform a robust analysis by varying the number of reference models and queries. The results remain largely consistent across different configurations, as shown in Appendix A.8.

Model-based TDD methods achieve generally better detection performance. Across datasets and target models, model-based detection algorithms consistently outperform other methods. For instance, as shown in Table 4, all five model-based algorithms achieve an average performance near or above 0.65 across all 12 datasets, whereas the performance of metric-based and learning-based methods is substantially lower. Overall, these results highlight the performance advantage of model-based TDD methods.

Data's task label information is useful. Some TDD methods leverage the focal data point's ground truth label (e.g., image class label or sentiment class), while others do not. Experimental results demonstrate that incorporating label information significantly improves detection performance. For instance, `Metric-ment` consistently outperforms `Metric-ent` by utilizing data labels. Similar improvements are observed with `Learn-label` compared to `Learn-original`, where the former benefits from leveraging the label information while the latter does not.

Hybrid method has potential. Notably, `Query-ref`, which generates crafted query data for the image modality, achieves the best performance among all 21 TDD algorithms. While categorized as query-based, this method also trains reference models, similar to model-based methods, making it a hybrid of query-based and model-based approaches. This highlights the potential of combining the merits of different methods to enhance detection accuracy.

Table 4: AUROC of TDD algorithms across different datasets. WRN28-2, Multilayer Perceptron, and DistilBERT are trained on image, tabular, and text datasets, respectively. The last column of each table displays the average performance of the corresponding TDD algorithm across different datasets. Complete results with standard deviations are provided in Table 21 in the Appendix A.9.

Modality	Image				Tabular				Text				Avg.
Dataset	CIFAR-10	CIFAR-100	BloodMNIST	CelebA	Purchase	Texas	Adult	Student	Rotten	Tweet	CoLA	ECtHR	
`Metric-loss`	**0.635**	**0.858**	**0.527**	**0.509**	0.619	0.629	0.500	**0.566**	**0.582**	**0.566**	**0.571**	0.521	0.590
`Metric-conf`	**0.635**	**0.858**	**0.527**	**0.509**	0.619	0.629	0.500	**0.566**	**0.582**	**0.566**	**0.571**	0.521	0.590
`Metric-corr`	0.552	0.708	0.517	0.507	0.551	0.610	**0.501**	0.560	0.557	0.550	0.550	0.519	0.557
`Metric-ent`	0.628	0.848	0.525	0.508	0.616	0.563	0.498	0.520	0.561	0.528	0.519	0.507	0.568
`Metric-ment`	**0.635**	**0.858**	**0.527**	**0.509**	**0.620**	**0.630**	0.500	**0.566**	**0.582**	**0.566**	**0.571**	**0.522**	**0.591**
`Learn-original`	0.631	0.870	0.508	0.503	0.652	0.597	0.502	0.531	0.558	0.529	0.568	0.506	0.580
`Learn-top3`	0.628	0.851	0.526	0.503	0.677	0.573	0.500	0.520	0.561	0.528	0.531	0.502	0.575
`Learn-sorted`	0.628	0.850	**0.529**	**0.508**	0.666	0.573	0.501	0.520	0.561	0.528	0.510	0.501	0.573
`Learn-label`	0.633	0.882	0.515	0.507	0.656	0.669	**0.503**	0.590	**0.584**	**0.570**	**0.622**	0.517	0.604
`Learn-merge`	**0.656**	**0.893**	0.523	0.507	**0.684**	**0.686**	0.502	**0.595**	**0.584**	0.569	0.620	**0.530**	**0.612**
`Model-loss`	0.664	0.852	**0.560**	**0.522**	0.725	**0.767**	**0.509**	**0.670**	**0.773**	**0.756**	**0.752**	**0.655**	**0.684**
`Model-calibration`	0.639	0.763	0.553	0.520	0.684	0.718	0.508	0.648	0.695	0.714	0.699	0.638	0.648
`Model-lira`	**0.690**	**0.937**	0.536	0.512	**0.755**	0.753	0.503	0.634	0.753	0.728	0.737	0.604	0.679
`Model-fpr`	0.647	0.852	0.552	0.516	0.697	0.723	0.507	0.641	0.679	0.722	0.708	0.635	0.657
`Model-robust`	0.635	0.889	0.552	0.520	0.711	0.762	**0.509**	0.669	0.766	0.745	0.746	0.621	0.677
`Query-augment`	0.573	0.761	0.517	0.502	0.612	**0.612**	**0.500**	0.560	0.570	0.551	0.561	0.518	0.570
`Query-transfer`	0.522	0.622	0.503	0.502	0.529	0.581	0.499	0.522	0.530	0.530	0.526	0.510	0.531
`Query-adv`	0.615	0.838	0.508	0.514	**0.620**	0.579	**0.500**	**0.563**	**0.571**	0.551	**0.568**	0.519	0.579
`Query-neighbor`	0.511	0.553	0.497	0.501	0.533	**0.612**	**0.500**	0.535	0.533	**0.556**	0.550	**0.522**	0.534
`Query-qrm`	0.532	0.574	0.510	0.505	0.523	0.530	**0.500**	0.526	0.524	0.521	0.511	0.512	0.522
`Query-ref`	**0.735**	**0.941**	**0.566**	**0.526**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	**0.692**

Table 5: AUROC of TDD algorithms across different target model architectures. MLP stands for Multilayer Perceptron, and LR stands for Logistic Regression. The last column of each table displays the average performance of the corresponding TDD algorithm across different model architectures. Complete results with standard deviations are provided in Table 22 in the Appendix A.9.

Dataset	CIFAR10(Image)				Purchase(Tabular)			Rotten-tomatoes(Text)			Avg.
Target model	WRN28-2	ResNet18	VGG11	MobileNet-v2	MLP	CatBoost	LR	DistilBERT	RoBERTa	Flan-T5	
`Metric-loss`	**0.635**	**0.659**	0.684	**0.592**	0.619	0.948	0.640	**0.582**	**0.571**	**0.517**	**0.645**
`Metric-conf`	**0.635**	**0.659**	0.684	**0.592**	0.619	0.948	0.640	**0.582**	**0.571**	**0.517**	**0.645**
`Metric-corr`	0.552	0.557	0.574	0.548	0.551	0.636	0.622	0.557	0.542	0.513	0.565
`Metric-ent`	0.628	0.654	0.680	0.582	0.616	0.943	0.594	0.561	0.555	0.509	0.632
`Metric-ment`	**0.635**	**0.659**	**0.685**	**0.592**	**0.620**	**0.950**	**0.642**	**0.582**	**0.571**	**0.517**	**0.645**
`Learn-original`	0.631	0.623	0.694	0.533	0.652	0.935	0.644	0.558	0.546	0.515	0.633
`Learn-top3`	0.628	0.653	0.680	**0.582**	0.677	0.967	0.660	0.561	0.555	0.509	0.647
`Learn-sorted`	0.628	**0.654**	0.680	0.578	0.666	0.963	0.661	0.561	0.555	0.509	0.646
`Learn-label`	0.633	0.612	0.707	0.557	0.656	0.954	0.701	**0.584**	0.565	**0.520**	0.649
`Learn-merge`	**0.656**	0.628	**0.727**	0.528	**0.684**	**0.968**	**0.716**	**0.584**	**0.566**	0.518	**0.657**
`Model-loss`	0.664	0.709	0.729	0.607	0.725	0.975	0.776	**0.773**	**0.656**	**0.602**	0.721
`Model-calibration`	0.639	0.671	0.690	0.595	0.684	0.865	0.719	0.695	0.622	0.592	0.677
`Model-lira`	**0.690**	**0.749**	**0.780**	0.601	**0.755**	**0.995**	0.761	0.753	0.630	0.569	**0.728**
`Model-fpr`	0.647	0.684	0.712	**0.619**	0.697	0.976	0.724	0.679	0.623	0.589	0.695
`Model-robust`	0.635	0.677	0.704	0.602	0.711	0.983	**0.796**	0.766	0.639	0.574	0.709
`Query-augment`	0.573	0.575	0.633	0.542	0.612	0.696	**0.664**	0.570	0.546	0.512	0.592
`Query-transfer`	0.522	0.522	0.533	0.507	0.529	0.587	0.574	0.530	0.515	0.506	0.533
`Query-adv`	0.615	0.621	0.666	0.583	**0.620**	0.727	0.662	**0.571**	**0.552**	**0.516**	0.613
`Query-neighbor`	0.511	0.512	0.509	0.509	0.533	0.820	0.530	0.533	0.527	0.504	0.549
`Query-qrm`	0.532	0.537	0.541	0.530	0.523	**0.946**	0.632	0.524	0.528	0.506	0.580
`Query-ref`	**0.735**	**0.800**	**0.843**	**0.656**	N/A	N/A	N/A	N/A	N/A	N/A	**0.759**

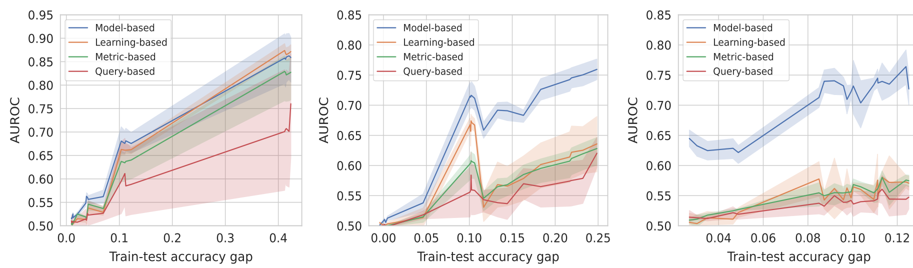
3.3 THE IMPACT OF TARGET MODEL

In this section, we examine the impact of the target model on TDD detection performance.

3.3.1 DATA MEMORIZATION AND OVERFITTING

A common view is that the effectiveness of TDD is closely tied to the level of training-data memorization or overfitting exhibited by the target model during training (Yeom et al., 2018; Long et al., 2018). The disparity between the target model's accuracy on the training set and the test set, known as the *train-test accuracy gap*, serves as an indicator of data memorization in prior literature (Carlini et al., 2022a). In our experiments, we document the train-test gaps of 12 distinct target models in Table 4, along with the corresponding performance of all detection methods. Specifically, the training of target models is repeated five times with different random training samples. Figure 2 illustrates the performance of various detection algorithms across different train-test gaps, with error bars representing 95% confidence intervals obtained from five independent trials. It is evident that the performance of all TDD methods is positively correlated with the train-test accuracy gap of the target model.

Takeaway. It is crucial for future advanced TDD algorithms to evaluate the generalizability of target models and report TDD performance when the train-test gap is small.



(a) Image (b) Tabular (c) Text

Figure 2: TDD algorithm performance (AUROC) versus the target model's train-test accuracy gap. The reported performance is averaged across all datasets.

3.3.2 Target model size

We examine the impact of target model size on TDD performance. Specifically, in our experiment, we vary the number of layers in the ResNet architecture for image data, the number of hidden units in the MLP for tabular data, and the parameter sizes of the large language model Pythia for text data.

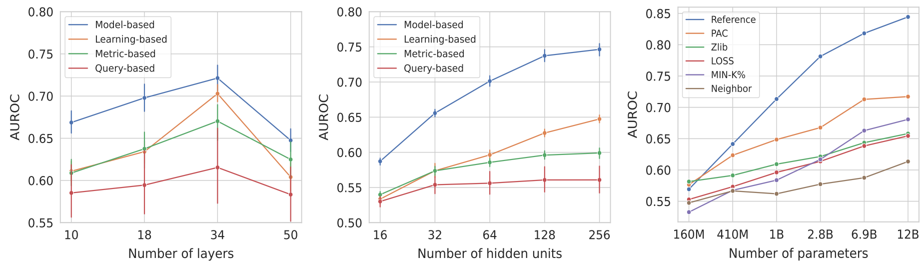
Due to limitations in computing resources, TDD on large models often does not involve the creation of shadow models or reference models. Drawing from prior studies (Shi et al., 2024; Duan et al., 2024), we utilize multiple detection methods suitable for pretrained large language models. Detailed descriptions of these detection methods can be found in Appendix A.2. Additionally, due to the lack of specific information regarding the training data used for large language models, we utilize the WIKIMIA (Shi et al., 2024), which collects training and non-training data for the large language model based on the model's release timeline to evaluate the TDD method in large language models.

The results of the experiments are illustrated in Figure 4. It is observed that, in most cases, the performance of the detection method improves as the size of the model increases. This aligns with the expectation that an increase in model size typically enhances model memorization (Carlini et al., 2023; Arpit et al., 2017). However, an exception occurs when the number of layers in the ResNet model is expanded from 34 to 50, resulting in a decline in the detection method's performance. One potential explanation for this anomaly is that the integration of residual connections in ResNet helps alleviate issues related to excessive memorization stemming from the increased depth of the model.

Takeaway. TDD algorithms generally demonstrate improved performance as the model size increases, highlighting their potential in the era of large models.

3.4 Performance when Knowledge about the Target Model is Limited

In the above experiments, we assumed that despite the black-box setting, TDD algorithms had some knowledge about the target model's training algorithm. However, in real-world scenarios, it is possible that the data owner may lack detailed knowledge about the target model's architecture, leading to significant differences between the reference and shadow models constructed by the TDD method and the actual target model. To explore this issue, we assess the performance of TDD when the reference and shadow models differ from the target model.



(a) Image:ResNet (b) Tabular:MLP (c) Text:Pythia

Figure 3: TDD algorithm performance (AUROC) versus model size, measured by the number of layers in ResNet, the number of hidden units in MLP, and the number of parameters in large language models Pythia.

The results, presented in Table 14, show a noticeable decline in detection performance when the data owner has limited knowledge about the target model's architecture. For example, `Learn-original` exhibits a 5.3% performance decline when using ResNet18 as the shadow model. This performance degradation can be attributed to discrepancies between the shadow and target models, which result in biased input features for training the auxiliary classifier.

Takeaway. The overall performance of TDD algorithms, without knowledge of the target model's training algorithm, remains unsatisfactory. This underscores the ineffectiveness of TDD algorithms on most datasets when information about the target model is limited.

Table 6: TDD algorithm performance (AUROC) when the reference or shadow models are different from the target model (i.e., when knowledge about the target model is limited). Complete results with standard deviations are provided in Table 23 in Appendix A.9.

Target model	WRN28-2(CIFAR-10)				MLP(Purchase)			DistilBERT(Rotten-tomatoes)			Avg.
Shadow/reference model	WRN28-2	ResNet18	VGG11	MobileNet-v2	MLP	CatBoost	LR	DistilBERT	RoBERTa	Flan-T5	
`Learn-original`	0.631	0.578	0.632	0.539	0.652	0.651	0.564	0.558	0.560	0.546	0.591
`Learn-top3`	0.628	0.628	0.628	0.628	0.677	0.646	0.515	0.561	0.561	0.561	0.603
`Learn-sorted`	0.628	**0.629**	0.628	**0.629**	0.666	**0.656**	0.532	0.561	0.561	0.561	**0.605**
`Learn-label`	0.633	0.591	0.644	0.563	0.656	0.651	0.551	**0.584**	**0.584**	**0.580**	0.604
`Learn-merge`	**0.656**	0.581	**0.651**	0.509	**0.684**	0.517	**0.595**	**0.584**	**0.584**	**0.580**	0.594
`Model-loss`	0.664	0.657	0.641	0.632	0.725	0.608	0.611	**0.773**	0.607	0.589	0.651
`Model-calibration`	0.639	0.634	0.617	0.614	0.684	0.579	0.588	0.695	0.595	0.587	0.623
`Model-lira`	**0.690**	0.659	**0.666**	0.610	**0.755**	**0.686**	0.588	0.753	0.602	0.553	**0.656**
`Model-fpr`	0.647	**0.668**	0.638	**0.664**	0.697	0.645	**0.643**	0.679	0.557	0.567	0.641
`Model-robust`	0.635	0.639	0.633	0.621	0.711	0.632	0.625	0.766	**0.624**	**0.591**	0.648
`Query-augment`	0.573	0.555	0.575	0.552	**0.612**	0.612	0.612	**0.570**	**0.569**	**0.565**	0.580
`Query-transfer`	0.522	0.529	0.518	0.518	0.529	0.535	0.529	0.530	0.529	0.514	0.525
`Query-qrm`	0.532	0.532	0.533	0.532	0.523	**0.625**	**0.622**	0.524	0.524	0.528	0.548
`Query-ref`	**0.735**	**0.740**	**0.722**	**0.708**	N/A	N/A	N/A	N/A	N/A	N/A	**0.726**

3.5 Performance tradeoff of TDD algorithms

Table 7: Quantitative evaluation of different types of TDD algorithms including the average and best AUROC, maximum runtime and memory usage.

Algorithm type	Average performance	Best performance	Running time(s)	Memory usage(MB)
Metric-based	0.626	0.645	232	0
Learning-based	0.646	0.657	2,107	855
Model-based	0.706	0.728	4,089	13,680
Query-based	0.604	0.759	40,963	13,680

Most evaluations of TDD algorithms primarily focus on detection accuracy. However, other factors, such as computational efficiency, are equally important in real-world applications. For instance, model-based methods, which require building numerous reference models, may be too costly in terms of time and memory when applied to large AI models. Therefore, in TDDBench, we emphasize the computational complexity of running different TDD algorithms. Specifically, we document the maximum runtime and memory usage for each type of TDD algorithm. This provides a holistic evaluation beyond detection accuracy. We present an overall assessment of the four types of TDD algorithms in Table 7. Evidently, each type of TDD algorithm has its own advantages and disadvantages. While model-based methods offer the best average performance, they come with significantly higher running time and memory usage. Therefore, data owners performing TDD must strike a balance between practicality, resource utilization, and detection accuracy, depending on their specific scenario. For instance, in resource-constrained environments, metric-based methods are a suitable choice for TDD, as they require minimal computational resources and fewer assumptions compared to other methods.

Takeaway. None of the TDD algorithms are satisfactory, as performance improvements often necessitate increased consumption of computing resources.

4 Related Work

Training data detection (TDD) is commonly employed to assess privacy risks in machine learning models (Murakonda & Shokri, 2020). It has been applied across various domains, including image classification (Hui et al., 2021), text generation (Shejwalkar et al., 2021), graph neural networks (Wu et al., 2021), and recommendation systems (Zhang et al., 2021). TDD has a wide range of applications such as dataset copyright protection (Maini et al., 2021) and for verifying machine unlearning (Chen et al., 2021). Shokri et al., 2017 introduce the first TDD algorithm, utilizing shadow models to help identifying differences in the model's predictions for training data versus other data. Yeom et al., 2018 demonstrate that satisfactory results could be achieved by utilizing only the loss of the target model on the sample. Carlini et al., 2022a criticize methods based solely on the target model's output, arguing that they overlook the inherent characteristics of the data, which can lead to biased estimates regarding whether a sample belongs to the training dataset. They propose training multiple reference models to better understand how the sample's characteristics influence metrics like loss. There is a rapidly growing body of literature on TDD methods, and we provide a brief summarization in Section 2.3.

Existing benchmarking works. He et al., 2022b evaluate 9 TDD algorithms on image data, while Niu et al., 2023 expand the evaluation to 15 algorithms, focusing on how sample differences within datasets affect TDD performance. Duan et al., 2024 investigate five TDD algorithms on large language models (LLMs) and find that current TDD algorithms perform poorly in this context. In summary, existing TDD benchmarks have limited coverage of data modalities and algorithms, underscoring the need for a more comprehensive analysis of TDD algorithms. This paper, along with the developed TDDBench, aims to address this gap by providing in-depth insights into the development and performance tradeoff in state-of-the-art TDD algorithms.

5 Conclusions

In this article, we introduce TDDBench, a novel and comprehensive training data detection benchmark. Unlike existing benchmarks, TDDBench extends evaluations across multiple data modalities, including image, tabular, and text. It also includes large language models and benchmarks 21 state-of-the-art TDD algorithms. Our comprehensive evaluation sheds critical light on the development of TDD algorithms and helps both researchers and practitioners reconsider the trade-offs involved in using TDD algorithms. For example, our evaluation shows that model-based TDD algorithms outperform others but at the cost of higher time and memory complexity. Additionally, all existing TDD algorithms experience performance degradation when the target model avoids overfitting. **Based on our findings with TDDBench, we believe future work on TDD algorithms should focus on, but not be limited to: (1) designing TDD algorithms that are robust against target models less prone to overfitting, (2) developing TDD algorithms that can effectively address defense countermeasures, (3) creating TDD algorithms that require minimal knowledge of the target model's architecture and data access, (4) achieving a better balance between performance and practical considerations such as computational complexity, and (5) tailoring algorithms to specific application contexts or training methods, such as training data detection for recommendation systems and semi-supervised models.**

REFERENCES

Alaa Abd-Alrazaq, Rawan AlSaad, Dari Alhuwail, Arfan Ahmed, Padraig Mark Healy, Syed Latifi, Sarah Aziz, Rafat Damseh, Sadam Alabed Alrazak, Javaid Sheikh, et al. Large language models in medical education: opportunities, challenges, and future directions. *JMIR Medical Education*, 9(1):e48291, 2023.

Guy Amit, Abigail Goldsteen, and Ariel Farkash. Sok: Reducing the vulnerability of fine-tuned language models to membership inference attacks. *arXiv preprint arXiv:2403.08481*, 2024.

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. In *International conference on machine learning*, pp. 233–242. PMLR, 2017.

Arthur Asuncion, David Newman, et al. Uci machine learning repository, 2007.

Kyungjune Baek and Hyunjung Shim. Commonality in natural images rescues gans: Pretraining gans with generic and privacy-free synthetic data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7854–7864, 2022.

Francesco Barbieri, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1644–1650, 2020.

Martin Bertran, Shuai Tang, Aaron Roth, Michael Kearns, Jamie H Morgenstern, and Steven Z Wu. Scalable membership inference attacks via quantile regression. *Advances in Neural Information Processing Systems*, 36, 2024.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *28th USENIX security symposium (USENIX security 19)*, pp. 267–284, 2019.

Nicholas Carlini, Florian Tramer, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Ulfar Erlingsson, et al. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2633–2650, 2021.

Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramer. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pp. 1897–1914. IEEE, 2022a.

Nicholas Carlini, Matthew Jagielski, Chiyuan Zhang, Nicolas Papernot, Andreas Terzis, and Florian Tramer. The privacy onion effect: Memorization is relative. *Advances in Neural Information Processing Systems*, 35:13263–13276, 2022b.

Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*, 2023.

Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 9650–9660, 2021.

Ilias Chalkidis, Nicolas Garneau, Cătălina Goanță, Daniel Katz, and Anders Søgaard. Lexfiles and legallama: Facilitating english multinational legal language model development. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 15513–15535, 2023.

Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. When machine unlearning jeopardizes privacy. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, pp. 896–911, 2021.

Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *International conference on machine learning*, pp. 1964–1974. PMLR, 2021.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Paulo Cortez and Alice Maria Gonçalves Silva. Using data mining to predict secondary school student performance. *Proceedings of 5th Annual Future Business Technology Conference*, 2008.

Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. Catboost: gradient boosting with categorical features support. *arXiv preprint arXiv:1810.11363*, 2018.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. Do membership inference attacks work on large language models? *arXiv preprint arXiv:2402.07841*, 2024.

Jamie Hayes, Luca Melis, George Danezis, and Emiliano De Cristofaro. Logan: Membership inference attacks against generative models. *arXiv preprint arXiv:1705.07663*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9729–9738, 2020.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 16000–16009, 2022a.

Xinlei He, Zheng Li, Weilin Xu, Cory Cornelius, and Yang Zhang. Membership-doctor: Comprehensive assessment of membership inference against machine learning models. *arXiv preprint arXiv:2208.10445*, 2022b.

David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*. John Wiley & Sons, 2013.

Bo Hui, Yuchen Yang, Haolin Yuan, Philippe Burlina, Neil Zhenqiang Gong, and Yinzhi Cao. Practical blind membership inference attack via differential comparisons. *arXiv preprint arXiv:2101.01341*, 2021.

Bargav Jayaraman, Lingxiao Wang, Katherine Knipmeyer, Quanquan Gu, and David Evans. Revisiting membership inference under realistic assumptions. *Proceedings on Privacy Enhancing Technologies*, 2021.

Jinyuan Jia, Ahmed Salem, Michael Backes, Yang Zhang, and Neil Zhenqiang Gong. Memguard: Defending against black-box membership inference attacks via adversarial examples. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 259–274, 2019.

Yigitcan Kaya and Tudor Dumitras. When does data augmentation help with membership inference attacks? In *International conference on machine learning*, pp. 5345–5355. PMLR, 2021.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Meghdad Kurmanji, Peter Triantafillou, Jamie Hayes, and Eleni Triantafillou. Towards unbounded machine unlearning. *Advances in neural information processing systems*, 36, 2024.

Klas Leino and Matt Fredrikson. Stolen memories: Leveraging model memorization for calibrated {White-Box} membership inference. In *29th USENIX security symposium (USENIX Security 20)*, pp. 1605–1622, 2020.

Zheng Li and Yang Zhang. Membership leakage in label-only exposures. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 880–895, 2021.

Hongbin Liu, Jinyuan Jia, Wenjie Qu, and Neil Zhenqiang Gong. Encodermi: Membership inference against pre-trained encoders in contrastive learning. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2081–2095, 2021a.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019. URL `http://arxiv.org/abs/1907.11692`.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 10012–10022, 2021b.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Yunhui Long, Vincent Bindschaedler, Lei Wang, Diyue Bu, Xiaofeng Wang, Haixu Tang, Carl A Gunter, and Kai Chen. Understanding membership inferences on well-generalized learning models. *arXiv preprint arXiv:1802.04889*, 2018.

Pratyush Maini, Mohammad Yaghini, and Nicolas Papernot. Dataset inference: Ownership resolution in machine learning. *arXiv preprint arXiv:2104.10706*, 2021.

Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schoelkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership inference attacks against language models via neighbourhood comparison. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 11330–11343, 2023.

Maximilian Mozes, Xuanli He, Bennett Kleinberg, and Lewis D Griffin. Use of llms for illicit purposes: Threats, prevention measures, and vulnerabilities. *arXiv preprint arXiv:2308.12833*, 2023.

Sasi Kumar Murakonda and Reza Shokri. Ml privacy meter: Aiding regulatory compliance by quantifying the privacy risks of machine learning. *arXiv preprint arXiv:2007.09339*, 2020.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Machine learning with membership privacy using adversarial regularization. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 634–646, 2018.

Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pp. 739–753. IEEE, 2019.

Jun Niu, Xiaoyan Zhu, Moxuan Zeng, Ge Zhang, Qingyang Zhao, Chunhui Huang, Yangming Zhang, Suyu An, Yangzhong Wang, Xinghui Yue, et al. Sok: Comparing different membership inference attacks with a comprehensive benchmark. *arXiv preprint arXiv:2307.06123*, 2023.

Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pp. 115–124, 2005.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*, 71(599-607):6, 1986.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Hervé Jégou. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning*, pp. 5558–5567. PMLR, 2019.

Ahmed Salem, Yang Zhang, Mathias Humbert, Pascal Berrang, Mario Fritz, and Michael Backes. Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models. In *Proceedings 2019 Network and Distributed System Security Symposium*. Internet Society, 2019.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.

Virat Shejwalkar, Huseyin A Inan, Amir Houmansadr, and Robert Sim. Membership inference attacks against nlp classification models. In *NeurIPS 2021 Workshop Privacy in Machine Learning*, 2021.

Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pp. 3–18. IEEE, 2017.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in neural information processing systems*, 33:596–608, 2020.

Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2615–2632, 2021.

Liwei Song, Reza Shokri, and Prateek Mittal. Privacy risks of securing machine learning models against adversarial examples. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 241–257, 2019.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Stacey Truex, Ling Liu, Mehmet Emre Gursoy, Lei Yu, and Wenqi Wei. Demystifying membership inference attacks in machine learning as a service. *IEEE transactions on services computing*, 14 (6):2073–2089, 2019.

Mariia Vladimirova, Federico Pavone, and Eustache Diemert. Fairjob: A real-world dataset for fairness in online systems. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *EMNLP 2018*, pp. 353, 2018.

Lauren Watson, Chuan Guo, Graham Cormode, and Alexandre Sablayrolles. On the importance of difficulty calibration in membership inference attacks. In *International Conference on Learning Representations*, 2021.

Yuxin Wen, Arpit Bansal, Hamid Kazemi, Eitan Borgnia, Micah Goldblum, Jonas Geiping, and Tom Goldstein. Canary in a coalmine: Better membership inference with ensembled adversarial queries. In *The Eleventh International Conference on Learning Representations*, 2023.

Bang Wu, Xiangwen Yang, Shirui Pan, and Xingliang Yuan. Adapting membership inference attacks to gnn for graph classification: Approaches and implications. In *2021 IEEE International Conference on Data Mining (ICDM)*, pp. 1421–1426. IEEE, 2021.

Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, and Bingbing Ni. Medmnist v2-a large-scale lightweight benchmark for 2d and 3d biomedical image classification. *Scientific Data*, 10(1):41, 2023.

Jiayuan Ye, Aadyaa Maddi, Sasi Kumar Murakonda, Vincent Bindschaedler, and Reza Shokri. Enhanced membership inference attacks against machine learning models. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3093–3106, 2022.

Wentao Ye, Jiaqi Hu, Liyao Li, Haobo Wang, Gang Chen, and Junbo Zhao. Data contamination calibration for black-box llms. *arXiv preprint arXiv:2405.11930*, 2024.

Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *2018 IEEE 31st computer security foundations symposium (CSF)*, pp. 268–282. IEEE, 2018.

Zuobin Ying, Yun Zhang, and Ximeng Liu. Privacy-preserving in defending against membership inference attacks. In *Proceedings of the 2020 Workshop on Privacy-Preserving Machine Learning in Practice*, pp. 61–63, 2020.

Sergey Zagoruyko. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Sajjad Zarifzadeh, Philippe Liu, and Reza Shokri. Low-cost high-power membership inference attacks. In *Forty-first International Conference on Machine Learning*, 2024.

Minxing Zhang, Zhaochun Ren, Zihan Wang, Pengjie Ren, Zhunmin Chen, Pengfei Hu, and Yang Zhang. Membership inference attacks against recommender systems. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 864–879, 2021.

Jie Zhu, Jirong Zha, Ding Li, and Leye Wang. A unified membership inference method for visual self-supervised encoder via part-aware capability. *arXiv preprint arXiv:2404.02462*, 2024.

A Appendix

A.1 Training data detection on vision transformer models

In this section, we showcase the performance of the TDD algorithm on vision transformer models. Specifically, we trained ViT (Dosovitskiy, 2020) and Swin (Liu et al., 2021b) models on the CIFAR-10 dataset and evaluated the TDD algorithm's effectiveness on these models. As shown in Table 8 and Table 9, the TDD algorithm remains effective on vision transformer models, and model-based algorithms continue to have clear advantages over other types of methods.

Table 8: TDD performance across different metrics on ViT (Dosovitskiy, 2020) trained on CIFAR-10 dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
Metric-loss	**0.555**	**0.811**	**0.659**	**0.583**	**0.190**	0.644	**0.167**	0.010	0.119	**0.599**
Metric-conf	**0.555**	**0.811**	**0.659**	**0.583**	**0.190**	0.644	**0.167**	0.010	0.119	**0.599**
Metric-corr	**0.555**	0.796	0.654	0.581	0.204	0.633	0.162	0.000	0.000	0.581
Metric-ent	0.536	0.499	0.517	0.536	0.501	**0.428**	0.071	**0.011**	0.114	0.543
Metric-ment	**0.555**	0.810	**0.659**	**0.583**	**0.190**	0.644	**0.167**	**0.011**	**0.121**	**0.599**
Learn-original	0.522	0.675	0.589	0.532	0.325	0.612	0.063	0.013	0.122	0.537
Learn-top3	0.539	0.477	0.506	0.536	0.523	0.405	0.072	0.010	0.116	0.541
Learn-sorted	0.539	0.474	0.504	0.536	0.526	**0.402**	0.072	0.010	0.116	0.541
Learn-label	0.552	0.787	0.649	0.577	0.213	0.634	0.153	0.015	**0.141**	0.597
Learn-merge	**0.556**	**0.799**	**0.655**	**0.583**	**0.201**	0.634	**0.165**	**0.016**	0.138	**0.604**
Model-loss	0.596	0.718	0.651	**0.617**	0.283	0.483	0.235	**0.056**	0.244	**0.672**
Model-calibration	0.578	**0.764**	**0.658**	0.606	**0.236**	0.553	0.212	0.046	0.222	0.653
Model-lira	0.562	0.735	0.637	0.584	0.265	0.567	0.168	0.052	0.219	0.631
Model-fpr	**0.611**	0.594	0.603	0.610	0.406	**0.374**	0.220	0.036	0.231	0.651
Model-robust	0.603	0.666	0.633	0.615	0.334	0.436	**0.231**	0.056	**0.255**	0.671
Query-augment	0.557	0.771	0.646	0.581	0.229	0.609	0.162	0.003	0.067	0.594
Query-transfer	0.525	0.738	0.614	0.538	0.262	0.662	0.077	0.009	0.102	0.538
Query-adv	0.564	0.793	0.659	0.593	0.207	0.608	0.186	0.008	0.119	0.590
Query-neighbor	0.504	0.393	0.442	0.505	0.607	0.383	0.010	0.001	0.061	0.506
Query-qrm	0.555	0.801	0.656	0.583	**0.199**	0.636	0.165	0.000	0.000	0.597
Query-ref	**0.676**	**0.558**	**0.611**	**0.649**	0.442	**0.260**	**0.298**	**0.089**	**0.308**	**0.718**

Table 9: TDD performance across different metrics on Swin (Liu et al., 2021b) trained on CIFAR-10 dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
Metric-loss	**0.571**	0.855	0.685	**0.609**	0.146	0.636	**0.218**	**0.009**	0.116	**0.621**
Metric-conf	**0.571**	0.855	0.685	**0.609**	0.146	0.636	**0.218**	**0.009**	0.116	**0.621**
Metric-corr	0.560	**0.915**	**0.695**	0.601	**0.085**	0.713	0.202	0.000	0.000	0.601
Metric-ent	0.547	0.690	0.611	0.562	0.310	**0.566**	0.124	**0.009**	0.115	0.573
Metric-ment	**0.571**	0.856	0.685	**0.609**	0.144	0.638	**0.218**	**0.009**	**0.117**	**0.621**
Learn-original	0.548	0.685	0.609	0.563	0.315	0.559	0.126	0.010	0.132	0.577
Learn-top3	0.553	0.625	0.587	0.562	0.375	0.501	0.124	0.009	0.116	0.573
Learn-sorted	0.553	0.624	0.586	0.562	0.376	**0.500**	0.124	0.009	0.116	0.573
Learn-label	0.571	**0.857**	**0.686**	**0.610**	**0.143**	0.638	0.219	**0.024**	0.169	0.638
Learn-merge	**0.574**	0.838	0.681	**0.610**	0.162	0.618	**0.220**	0.020	**0.172**	**0.643**
Model-loss	**0.623**	0.693	0.656	**0.639**	0.307	0.416	**0.278**	0.076	0.275	0.704
Model-calibration	0.581	**0.810**	**0.676**	0.615	**0.190**	0.580	0.230	0.061	0.224	0.668
Model-lira	0.590	0.772	0.669	0.621	0.228	0.531	0.241	**0.083**	0.278	0.686
Model-fpr	0.615	0.636	0.625	0.621	0.364	**0.394**	0.242	0.058	0.275	0.670
Model-robust	0.606	0.760	0.674	0.635	0.240	0.489	0.271	**0.083**	**0.300**	**0.705**
Query-augment	0.567	0.847	0.679	0.603	0.153	0.640	0.206	0.009	0.019	0.618
Query-transfer	0.542	0.850	0.662	0.570	0.150	0.711	0.139	0.009	0.105	0.560
Query-adv	0.577	**0.932**	0.713	0.627	**0.068**	0.677	0.254	0.008	0.155	0.645
Query-neighbor	0.505	0.422	0.460	0.506	0.578	**0.410**	0.012	0.002	0.079	0.507
Query-qrm	0.567	0.866	0.686	0.606	0.134	0.655	0.211	0.000	0.000	0.623
Query-ref	**0.639**	0.842	**0.726**	**0.689**	0.158	0.464	**0.378**	**0.128**	**0.383**	**0.759**

A.2 Training data detection algorithms in large language models

In this section, we present the performance of the TDD algorithm on various sizes of Llama models (Touvron et al., 2023). The experimental results in Table 10 indicate that the performance of the TDD algorithm improves as the model size increases, which aligns with the results observed for the detection results on Pythia (corresponds to Figure 4 in Section 3.3.2) . Additionally, we offer a brief introduction to the TDD algorithms for large language models; for more detailed information, please refer to the related works.

Table 10: The performance of the TDD algorithm across different sizes of Llama models.

Algorithm	Target Model			
	Llama-7b	Llama-13b	Llama-30b	Llama-65b
Neighbor	0.555	0.552	0.566	0.586
LOSS	0.666	0.678	0.704	0.707
PAC	0.679	0.689	0.704	0.714
Zlib	0.683	0.697	0.718	0.721
MIN-K%	0.697	0.715	0.737	0.737
Reference	**0.802**	**0.809**	**0.833**	**0.831**

Loss (Yeom et al., 2018) refers to the `Metric-loss` mentioned in the article. Instead of using cross-entropy as in classification models, the log likelihood of each text under the target model serves as the basis for detection in pretraining language models.

Zlib (Carlini et al., 2021) calibrates the sample's loss under the target model using the sample's zlib compression size.

MIN-K% (Shi et al., 2024) utilizes the k% of tokens with the lowest likelihoods as the detection basis, rather than average loss.

Reference (Carlini et al., 2021) borrows from model-based approaches, utilizing the reference model to help correct the detection basis derived from the prediction results of the target model. Reference models for TDD in large language models are typically open-source and have architectures similar to the target model, thus avoiding the significant computational cost of training the reference model from scratch.

Neighbor, or `Query-neighbor` (Mattern et al., 2023), supplements the detection information provided by the sample point x with the loss of the target model on the neighboring samples of x.

PAC, short for Polarized Augment Calibration (Ye et al., 2024), introduces a new detection metric called polarized distance through data augmentation. This metric helps determine whether data has been trained by large language models.

A.3 More discussion regarding the evaluated data in TDDBench

A.3.1 Training data detection performance across large-scale datasets

Table 11: Large-scale datasets used in TDDBench.

Dataset	#Samples	#Classes	Brief description
ImageNet-1K (Russakovsky et al., 2015)	1,331,167	1000	General dataset
FairJob (Vladimirova et al.)	1,072,226	2	Click prediction

In this section, we demonstrate the performance of the TDD algorithm on two large-scale datasets: ImageNet-1K (Russakovsky et al., 2015) and FairJob (Vladimirova et al.). ImageNet-1K is widely used for evaluating image classification models, while FairJob is designed to learn click prediction models and assess prediction bias between different gender groups. The statistics of these datasets are as follows.

Table 12: TDD performance across different metrics on WRN28-2 trained on ImageNet-1K dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
Metric-loss	0.570	**0.693**	0.626	0.586	**0.307**	0.522	0.171	0.009	0.143	0.608
Metric-conf	0.570	**0.693**	0.626	0.586	**0.307**	0.522	0.171	0.009	0.143	0.608
Metric-corr	**0.601**	0.445	0.511	0.575	0.555	**0.295**	0.150	0.000	0.000	0.575
Metric-ent	0.534	0.341	0.416	0.521	0.659	0.298	0.043	0.009	0.113	0.521
Metric-ment	0.572	0.692	**0.627**	**0.588**	0.308	0.517	**0.175**	**0.010**	**0.145**	**0.610**
Learn-original	0.514	0.447	0.478	0.512	0.553	0.424	0.024	0.000	0.116	0.509
Learn-top3	0.540	0.357	0.430	0.526	0.643	0.305	0.053	0.009	0.120	0.526
Learn-sorted	0.543	0.335	0.414	0.527	0.665	0.282	0.053	**0.010**	0.125	0.526
Learn-label	0.510	0.182	0.269	0.504	0.818	**0.175**	0.007	0.009	0.096	0.495
Learn-merge	**0.578**	**0.676**	**0.623**	**0.591**	**0.324**	0.494	**0.182**	**0.010**	**0.150**	**0.613**
Model-loss	**0.631**	0.699	**0.664**	**0.645**	0.301	0.409	**0.291**	**0.056**	**0.300**	**0.704**
Model-calibration	0.612	**0.719**	0.661	0.632	**0.281**	0.455	0.263	0.055	0.267	0.689
Model-lira	0.616	0.663	0.639	0.625	0.337	**0.413**	0.250	0.049	0.264	0.676
Model-fpr	0.622	0.698	0.657	0.637	0.302	0.425	0.273	0.033	0.273	0.685
Model-robust	0.622	0.701	0.659	0.637	0.299	0.426	0.275	0.050	0.284	0.694
Query-augment	0.575	0.491	0.529	0.564	0.509	0.363	0.128	0.013	0.142	0.562
Query-transfer	0.589	0.421	0.491	0.563	0.579	0.294	0.127	0.013	0.142	0.565
Query-adv	0.580	0.442	0.502	0.576	0.558	**0.290**	0.153	0.008	0.152	0.580
Query-neighbor	0.532	0.551	0.542	0.533	0.449	0.485	0.066	0.000	0.000	0.537
Query-qrm	0.572	**0.702**	**0.630**	0.587	**0.298**	0.529	0.173	0.000	0.000	0.616
Query-ref	**0.635**	0.558	0.594	**0.620**	0.442	0.319	**0.240**	**0.032**	**0.199**	**0.658**

Table 13: TDD performance across different metrics on MLP trained on FairJob dataset.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
Metric-loss	0.521	0.219	0.309	0.509	0.781	0.201	0.018	**0.010**	0.097	**0.500**
Metric-conf	0.521	0.219	0.309	0.509	0.781	0.201	0.018	**0.010**	0.097	**0.500**
Metric-corr	0.000	0.000	0.000	0.500	1.000	**0.000**	0.000	0.000	0.000	**0.500**
Metric-ent	**0.523**	**0.224**	**0.314**	**0.510**	**0.776**	0.204	**0.020**	**0.010**	**0.098**	**0.500**
Metric-ment	**0.523**	0.222	0.312	**0.510**	0.778	0.203	0.019	**0.010**	0.097	**0.500**
Learn-original	**0.525**	0.190	0.279	**0.509**	0.810	**0.173**	0.018	0.013	0.094	**0.500**
Learn-top3	0.520	**0.217**	**0.306**	0.508	**0.783**	0.200	0.016	0.004	0.017	**0.500**
Learn-sorted	0.521	0.209	0.298	0.508	0.791	0.192	0.016	0.012	0.095	**0.500**
Learn-label	0.519	0.214	0.303	0.508	0.786	0.199	0.015	0.010	0.019	0.499
Learn-merge	0.523	0.213	0.303	**0.509**	0.787	0.194	**0.019**	**0.013**	**0.102**	**0.500**
Model-loss	0.509	**0.475**	**0.492**	0.509	**0.525**	0.458	0.017	**0.013**	0.108	0.507
Model-calibration	0.526	0.262	0.350	**0.513**	0.738	0.236	**0.026**	0.011	**0.116**	**0.509**
Model-lira	0.502	0.412	0.453	0.502	0.588	0.408	0.004	0.011	0.098	0.490
Model-fpr	**0.530**	0.105	0.176	0.506	0.895	**0.093**	0.012	0.012	0.110	0.502
Model-robust	0.000	0.000	0.000	0.500	1.000	0.000	0.000	0.000	0.000	0.500
Query-augment	0.530	0.007	0.014	0.500	0.993	0.006	0.001	**0.007**	0.007	0.500
Query-transfer	**0.583**	0.004	0.008	0.501	0.996	0.003	0.001	0.004	**0.073**	0.492
Query-adv	0.000	0.000	0.000	0.500	1.000	**0.000**	0.000	0.000	0.000	0.500
Query-neighbor	0.518	0.228	0.317	**0.508**	0.772	0.213	**0.016**	0.005	0.071	**0.507**
Query-qrm	0.501	**0.993**	**0.666**	0.500	**0.007**	0.993	0.000	0.000	0.000	0.500

Specifically, we train a WRN28-2 model with ImageNet-1K and an MLP on the FairJob dataset as the target models, and record the detection performance of the TDD algorithm on these two models. As shown in Table 12 and Table 13, the performance of the TDD algorithm on large-scale datasets is not ideal. Notably, the algorithm achieves only about 50% AUROC on the FairJob dataset, indicating almost no detection capability. One possible explanation is that the large-scale dataset enhances the target model's generalization ability, reducing the gap between the training and test sets and weakening the TDD algorithm's performance.

A.3.2 Training data detection with different split rules

In the main experiment of our paper, we align with most existing literatures on TDD by assuming that the detection algorithm has access to the entire target dataset for training both the reference model and the shadow model. However, in this section, we impose constraints on the data access permissions available to the detection algorithm. We assume that the dataset used to train the reference and shadow models, which we refer to as the reference dataset, is derived from different split rules. Specifically, we consider four types of data access permissions for a detector: (1) The detector can access the entire target dataset, consistent with our initial experimental setup. (2) The detector can access only a portion (50%) of the target dataset. (3) The detector cannot access the target dataset but is aware of its data distribution. It can obtain reference data, which does not intersect with the target dataset, for use with the TDD algorithm. (4) The reference dataset is from a biased distribution, with the majority (80%) from half of the target dataset's categories and a minority (20%) from the other half.

We evaluated the performance of various TDD detection algorithms under these four scenarios. Notably, some algorithms, like Model-loss, only function under the first assumption. Others, such as metric-based methods, are designed to operate without relying on the reference dataset and are unaffected by different data partitioning rules. Our experimental results indicate that data partitioning significantly impacts the performance of TDD algorithms. Specifically, when the detector only has access to a biased data distribution, the performance of the TDD algorithm is minimized.

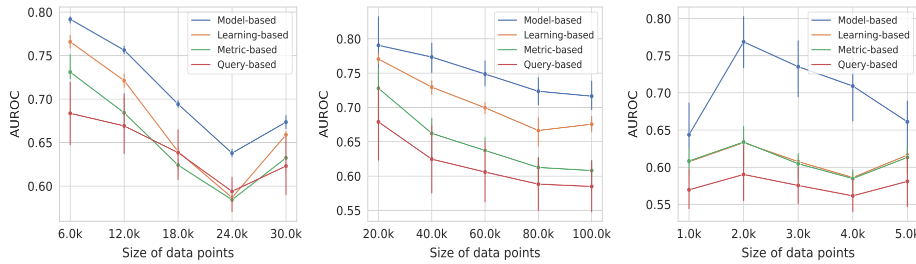
Future direction. A promising research direction is to explore methods to enhance TDD algorithm performance when the detector's data access is limited.

Table 14: TDD algorithm performance (AUROC) with different split rules. Reference data access ranges from 1 to 4, indicating the highest to lowest data permissions. 'N/A' indicates that the corresponding TDD algorithm is not applicable to the related split rule, while '-' indicates that the TDD algorithm is not affected by the split rule.

Dataset	CIFAR10(Image)				Purchase(Tabular)				Rotten-tomatoes(Text)			
Reference data access	1	2	3	4	1	2	3	4	1	2	3	4
Metric-loss	**0.635**	-	-	-	**0.620**	-	-	-	**0.582**	-	-	-
Metric-conf	**0.635**	-	-	-	**0.620**	-	-	-	**0.582**	-	-	-
Metric-corr	0.552	-	-	-	0.551	-	-	-	0.557	-	-	-
Metric-ent	0.628	-	-	-	0.616	-	-	-	0.561	-	-	-
Metric-ment	**0.635**	-	-	-	**0.620**	-	-	-	**0.582**	-	-	-
Learn-original	0.631	0.606	0.635	0.613	0.652	0.654	0.656	0.649	0.558	0.572	0.570	0.496
Learn-top3	0.628	**0.648**	0.651	0.645	0.677	0.683	0.687	0.633	0.561	0.575	0.572	0.428
Learn-sorted	0.628	**0.648**	0.651	**0.646**	0.667	0.645	0.686	**0.671**	0.561	0.575	0.572	**0.543**
Learn-label	0.634	0.606	0.658	0.613	0.656	0.668	0.663	0.654	**0.584**	0.593	0.590	0.557
Learn-merge	**0.656**	0.594	**0.667**	0.635	**0.684**	**0.689**	**0.689**	0.659	**0.584**	**0.594**	**0.593**	0.409
Model-loss	0.664	N/A	N/A	N/A	0.725	N/A	N/A	N/A	**0.773**	N/A	N/A	N/A
Model-calibration	0.639	0.601	0.657	0.596	0.684	0.640	0.680	0.621	0.695	0.673	**0.657**	**0.594**
Model-lira	**0.690**	N/A	N/A	N/A	**0.755**	N/A	N/A	N/A	0.753	N/A	N/A	N/A
Model-fpr	0.647	**0.634**	**0.666**	**0.626**	0.697	**0.677**	**0.684**	**0.638**	0.679	**0.699**	0.655	0.586
Model-robust	0.635	N/A	N/A	N/A	0.711	N/A	N/A	N/A	0.766	N/A	N/A	N/A
Query-augment	0.573	**0.568**	**0.577**	**0.586**	0.612	**0.612**	**0.613**	**0.613**	0.570	**0.574**	**0.578**	**0.581**
Query-transfer	0.522	0.532	0.506	0.508	0.529	0.526	0.528	0.520	0.530	0.519	0.515	0.507
Query-adv	0.615	-	-	-	**0.620**	-	-	-	**0.571**	-	-	-
Query-neighbor	0.511	-	-	-	0.533	-	-	-	0.533	-	-	-
Query-quantile	0.532	-	-	-	0.523	-	-	-	0.524	-	-	-
Query-canary	**0.735**	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

A.3.3 Training data detection with different sizes of data points

To examine the effect of the size of evaluated data points on the TDD algorithm, we varied the size of the target dataset and assessed the algorithm's performance on target models trained with these different dataset sizes. As shown in Figure 1, the model-based method consistently delivers the best detection performance across various sizes, which is consistent with earlier findings. Furthermore, there is no strong correlation between data size and the performance of the TDD algorithm.



(a) Image:CIFAR10 (b) Tabular:Purchase (c) Text:Rotten-tomatoe

Figure 4: TDD algorithm performance (AUROC) versus data size, measured by the number of data points in the target dataset.

A.4 Training data detection across different training methods

Research on TDD algorithms has primarily concentrated on two types of training methods. The first is supervised learning, which forms the basis for most TDD algorithms, covering various fields such as image (Carlini et al., 2022a), table (Shokri et al., 2017), and text (Amit et al., 2024). This is also the setting for our main experiments. The second type is self-supervised learning, which typically focuses on detecting whether the pre-trained corpus of large language models can be identified. This type of algorithm is also known as pretraining data detection (Shi et al., 2024), and our experiments on Llama and Pythia evaluated the TDD algorithm's performance in this setting.

Table 15: TDD performance on WRN28-2 trainied with semi-supervised method FixMatch (Sohn et al., 2020).

Dataset	CIFAR-10				CIFAR-100			
Algorithm	Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
Metric-loss	0.624	0.560	0.805	0.661	0.714	0.598	0.721	0.653
Metric-conf	0.624	0.560	0.805	0.661	0.714	0.598	0.721	0.653
Metric-corr	0.533	0.487	**0.910**	0.635	0.683	0.531	0.786	0.634
Metric-ent	**0.643**	**0.567**	0.870	**0.687**	**0.736**	0.587	**0.820**	**0.684**
Metric-ment	0.624	0.561	0.804	0.661	0.714	**0.602**	0.712	0.652
Learn-original	0.641	0.564	**0.889**	**0.690**	**0.737**	0.585	**0.832**	**0.687**
Learn-top3	**0.643**	**0.569**	0.862	0.685	0.733	0.591	0.801	0.680
Learn-sorted	**0.643**	0.568	0.865	0.686	0.734	0.591	0.804	0.681
Learn-label	0.617	0.550	0.835	0.664	0.722	0.584	0.778	0.668
Learn-merge	0.621	0.560	0.789	0.655	0.717	**0.607**	0.712	0.655
Model-loss	**0.620**	**0.561**	0.775	0.651	0.721	0.606	0.726	0.661
Model-calibration	0.604	0.546	0.774	0.641	0.690	0.555	0.737	0.633
Model-lira	0.614	0.555	0.778	0.648	0.720	**0.628**	0.685	0.655
Model-fpr	0.597	0.555	0.664	0.605	0.694	0.613	0.625	0.619
Model-robust	0.589	0.526	**0.868**	**0.655**	**0.726**	0.605	**0.745**	**0.668**
Query-augment	0.575	0.515	**0.888**	**0.652**	0.694	0.576	0.697	0.631
Query-transfer	0.531	0.496	0.589	0.539	0.595	0.453	0.699	0.550
Query-adv	0.607	0.561	0.753	0.643	**0.749**	**0.649**	**0.769**	**0.704**
Query-neighbor	0.505	0.473	0.582	0.522	0.515	0.393	0.470	0.428
Query-qrm	0.611	**0.562**	0.710	0.628	0.697	0.595	0.671	0.631
Query-ref	**0.642**	0.540	0.735	0.623	0.738	0.548	0.711	0.619

In this part, we investigate whether the TDD algorithm can be applied to different training algorithms, focusing on image datasets due to the scarcity of relevant studies. Specifically, we conduct experiments on CIFAR-10 and CIFAR-100 to explore the suitability of TDD algorithms for

Table 16: TDD performance on self-supervised image models: MAE, DINO, and MOCO.

Self-supervised model	Algorithm	CIFAR-10				CIFAR-100			
		Accuracy	Precision	Recall	F1-score	Accuracy	Precision	Recall	F1-score
MAE	Variance-onlyMI	0.515	0.516	0.492	0.504	0.517	0.517	0.540	0.528
	EncoderMI	0.532	0.532	0.611	0.569	0.517	0.522	0.499	0.510
	PartCrop	**0.577**	**0.576**	**0.640**	**0.606**	**0.584**	**0.577**	**0.600**	**0.589**
DINO	Variance-onlyMI	0.588	0.585	0.608	0.596	0.507	0.506	**0.648**	**0.568**
	EncoderMI	**0.664**	**0.656**	**0.636**	**0.646**	0.555	0.572	0.448	0.503
	PartCrop	0.591	0.607	0.538	0.570	**0.606**	**0.669**	0.411	0.509
MOCO	Variance-onlyMI	0.498	0.498	0.573	0.533	0.509	0.509	0.561	0.533
	EncoderMI	0.608	0.588	**0.722**	0.648	0.573	0.581	0.591	0.586
	PartCrop	**0.788**	**0.865**	0.669	**0.755**	**0.772**	**0.829**	**0.678**	**0.746**

semi-supervised and self-supervised learning, in addition to supervised learning. Notably, to our knowledge, there are currently no studies using the TDD algorithm for unsupervised learning on image datasets. We evaluated the TDD performance on WRN28-2 trained with the semi-supervised method FixMatch, as well as three self-supervised image models: MAE (He et al., 2022a), DINO (Caron et al., 2021), and MOCO (He et al., 2020). Similar to its application on large language models, using TDD on self-supervised image models requires designing specialized algorithms. Based on previous work (Zhu et al., 2024), we assessed the detection performance of Variance-onlyMI (Choquette-Choo et al., 2021), EncoderMI (Liu et al., 2021a), and PartCrop (Zhu et al., 2024) on these models. For more details about these algorithms, please refer to the relevant paper.

The experimental results lead to the following conclusions: 1) The TDD detection method remains effective in semi-supervised training, but its performance declines compared to supervised learning. Specifically, the model-based method, which shows clear advantages in supervised learning, performs moderately in the semi-supervised setting. This may be because the model-based approach relies on training a reference model, and its performance is significantly affected when the reference model is unaware of the semi-supervised learning method used by the target model. 2) Few TDD algorithms are suited for self-supervised training methods, and their performance is not ideal.

Future direction. Based on these experiments, we believe that studying TDD algorithms for specific training methods, particularly semi-supervised and self-supervised methods, is of great interest.

A.5 Defense strategies against training data detection

In the field of computer security, training data detection is known as a Membership Inference Attack, which aims to extract private information about the training data from target models. To counteract this detection, various measures (Baek & Shim, 2022; Ying et al., 2020) have been proposed. Since the effectiveness of training data detection is often linked to the degree of overfitting in the target model, many defense methods focus on reducing overfitting. These methods include dropout strategies (Salem et al., 2019), label smoothing (Kaya & Dumitras, 2021), early stopping (Song & Mittal, 2021), and data augmentation (Kaya & Dumitras, 2021).

Beyond reducing model overfitting, another key defense strategy involves modifying the output vector of the target model to lower the risk of training data leakage. For instance, Jia et al., 2019 suggests adding carefully designed noise to the model's output vector, which does not affect the target model's performance but can mislead detection algorithms. Shokri et al., 2017 recommends constraining the target model to output only prediction labels without confidence scores, rendering many TDD algorithms ineffective.

In addition to these common methods applicable across different data types, Hayes et al., 2017 employs differential privacy to prevent external parties from determining whether specific data was used in a generative model. Their results indicate that differential privacy can balance model usability with defense effectiveness. Shejwalkar et al., 2021 proposed a defense strategy based on knowledge distillation, demonstrating that the distilled model can better resist training data detection.

To better assess the robustness of TDD algorithms, we examine their performance when the target model is combined with various defense strategies. Specifically, we selected four general defense strategies: using dropout and label smoothing on the target model to mitigate overfitting, and altering the target model's output vector to noise vectors and hard label. Our experimental results across three datasets indicate that these defense strategies, particularly the addition of noise, can effectively diminish the performance of TDD algorithms. TDD algorithms with strong performance, such as those that are learning-based and model-based, heavily depend on the authenticity of the model's output vectors. Introducing small amounts of noise to the model output can significantly compromise the effectiveness of these TDD algorithms.

Future direction. Based on these findings, we suggest that to counter potential defense mechanism of the target model, a promising direction is to develop adaptive TDD approaches, which involve designing more effective TDD algorithms tailored to specific defense strategies.

Table 17: TDD performance under different defense strategies.

Dataset	CIFAR10(Image)					Purchase(Tabular)					Rotten-tomatoes(Text)				
Defense	None	Dropout	Smooth	Noise	Label-only	None	Dropout	Smooth	Noise	Label-only	None	Dropout	Smooth	Noise	Label-only
Metric-loss	**0.635**	**0.557**	**0.589**	**0.558**	N/A	**0.620**	**0.615**	**0.657**	**0.623**	N/A	**0.582**	**0.577**	**0.590**	**0.587**	N/A
Metric-conf	**0.635**	**0.557**	**0.589**	**0.558**	N/A	**0.620**	**0.615**	**0.657**	**0.623**	N/A	**0.582**	**0.577**	**0.590**	**0.587**	N/A
Metric-corr	0.552	0.547	0.557	0.549	**0.552**	0.551	0.558	0.556	0.551	**0.551**	0.557	0.546	0.555	0.554	**0.557**
Metric-ent	0.628	0.543	0.568	0.543	N/A	0.616	0.606	0.655	0.620	N/A	0.561	0.560	0.573	0.570	N/A
Metric-ment	**0.635**	**0.557**	**0.589**	**0.558**	N/A	**0.620**	**0.615**	0.656	**0.623**	N/A	**0.582**	**0.577**	**0.590**	**0.587**	N/A
Learn-original	0.631	0.540	0.493	0.539	N/A	0.652	0.617	0.605	0.659	N/A	0.558	0.558	0.574	0.572	N/A
Learn-top3	0.628	0.543	0.593	0.543	N/A	0.677	0.616	0.652	0.680	N/A	0.561	0.560	0.573	0.570	N/A
Learn-sorted	0.628	0.543	0.576	0.543	N/A	0.667	0.615	0.651	0.678	N/A	0.561	0.560	0.573	0.570	N/A
Learn-label	0.634	0.557	0.565	0.554	N/A	0.656	0.629	0.635	0.662	N/A	0.584	**0.578**	0.591	0.588	N/A
Learn-merge	**0.656**	**0.559**	**0.609**	**0.559**	N/A	**0.684**	**0.630**	**0.664**	**0.690**	N/A	**0.584**	0.577	**0.593**	**0.592**	N/A
Model-loss	0.664	0.606	0.614	0.607	N/A	0.725	0.693	0.732	0.726	N/A	**0.773**	**0.696**	0.677	**0.731**	N/A
Model-calibration	0.639	0.596	0.582	0.592	N/A	0.684	0.671	0.717	0.685	N/A	0.695	0.647	0.624	0.665	N/A
Model-lira	**0.690**	0.564	0.561	0.568	N/A	**0.755**	**0.698**	0.703	**0.756**	N/A	0.753	0.683	0.678	0.721	N/A
Model-fpr	0.647	0.571	0.518	0.568	N/A	0.697	0.651	0.547	0.703	N/A	0.679	0.653	0.537	0.675	N/A
Model-robust	0.635	**0.611**	**0.642**	**0.613**	N/A	0.711	0.692	**0.761**	0.712	N/A	0.766	0.692	**0.708**	0.730	N/A
Query-augment	0.511	0.551	0.586	0.547	0.511	0.533	0.602	0.619	0.612	0.533	0.533	0.559	0.571	0.571	0.533
Query-transfer	0.573	0.534	0.522	0.524	**0.573**	0.612	0.533	0.534	0.529	**0.612**	0.570	0.531	0.521	0.526	**0.570**
Query-adv	0.522	0.548	0.591	0.539	0.522	0.529	0.607	0.629	0.623	0.529	0.530	0.559	0.571	0.570	0.530
Query-neighbor	0.615	0.494	0.481	0.515	N/A	**0.620**	0.532	0.543	0.534	N/A	**0.571**	0.523	0.537	0.529	N/A
Query-qrm	0.532	0.575	0.633	**0.570**	N/A	0.523	**0.617**	**0.658**	**0.650**	N/A	0.524	**0.574**	**0.590**	**0.589**	N/A
Query-ref	**0.735**	**0.604**	**0.750**	0.497	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

A.6 DETAILS OF THE ALGORITHMS INCLUDED IN TDDBENCH

A.6.1 METRIC-BASED DECTECTION

Several studies (Carlini et al., 2019; 2021) indicate that models retain a certain degree of memory regarding the training data during the learning process. This memorization can result in significant differences between the model's predictions on training and test data, which can be leveraged as a decision basis for TDD algorithms. Specifically, **`Metric-loss`** (Yeom et al., 2018) utilizes the loss of the target model's prediction on data points as the detection criterion. Since the target model is instructed to minimize training loss during optimization, a training data point typically exhibits a lower loss than a test data point. Similarly, **`Metric-conf`** (`Metric-conf`idence) (Song et al., 2019) identifies that the maximum confidence of the target model's predictions can also serve as the detection criterion. **`Metric-corr`** (`Metric-corr`ectness) (Leino & Fredrikson, 2020) further demonstrates that even without access to the model's prediction confidence or logits for a specific data point, comparing the predicted label with the true label can provide an effective detection basis. `Metric-corr` achieves training data detection (TDD) with fewer assumptions than both `Metric-loss` and `Metric-conf`.

Beyond individual prediction values, the distribution of prediction results can also serve as a valuable detection criterion. **`Metric-ent`** (`Metric-ent`ropy) (Shokri et al., 2017; Song & Mittal, 2021) posits that the target model exhibits greater confidence in its predictions for training data, as evidenced by a more concentrated distribution of prediction confidences across different classes. Building on this, entropy is utilized as the detection criterion for `Metric-ent`. **`Metric-ment`** (Metric-modified entropy) (Song & Mittal, 2021) further incorporates the true label of the data point into `Metric-ent` to prevent the detection algorithm from predicting data points where the target model has misclassified as its training data.

A.6.2 LEARNING-BASED DECTECTION

The metric-based algorithms design various metrics to extract detection basis from the prediction results of the target model. However, manually designed metrics may not accurately capture the differences between the predicted results of training and test data. A more robust approach is to use neural networks to automatically extract training data detection (TDD)-friendly information from the target model's predictions, known as learning-based detection algorithms. **Learn-original** (Shokri et al., 2017) is the first algorithm to propose building an auxiliary classifier for TDD. It inputs the prediction vector of the target model into the auxiliary classifier, aiming to directly produce a detection result. To train the auxiliary classifier, `Learn-original` employs a shadow model similar to the target model, utilizing shadow training techniques. Since the shadow model's training process is conducted by the detectors, they can obtain both the training and test data of the shadow model. The predictions made by the shadow model on its training and test data are utilized to facilitate the training of the auxiliary classifier.

Different learning-based detection algorithms primarily differ in the input features of their auxiliary classifiers. For instance, **Learn-top3** and **Learn-sorted** (Salem et al., 2019) utilize the top-3 prediction confidences and ranked prediction vectors as input features, respectively. Building on `Learn-original`, **Learn-label** (Nasr et al., 2018) supplements the input features with the true label of the data point. **Learn-merge** (Amit et al., 2024) further incorporates the entropy, loss, and predicted label into the input features. It is noteworthy that while `Learn-original` builds multiple shadow models, most learning-based methods utilize only one. Moreover, previous work demonstrates that training data detection (TDD) with a single shadow model achieves performance comparable to that of multiple shadow models. Therefore, to ensure a fair comparison among learning-based methods, we standardize the number of shadow models to one for all learning-based approaches.

A.6.3 MODEL-BASED DECTECTION

The two lines of TDD methods discussed above rely solely on the prediction results of the target model, overlooking the inherent characteristics of the data points, which may introduce bias into the detection criteria. For example, abnormal training data may exhibit higher losses than normal test data due to inherent data characteristics (Carlini et al., 2022b;a), making it challenging for `Metric-loss` to detect these data. Therefore, the design of the detection criterion must consider data characteristics to eliminate bias.

Model-based algorithms address this issue by utilizing a set of reference models that share a similar architecture to the target model. These reference models are used to obtain predictions for data point x across different models, which helps to de-bias the detection criteria of the target model. To elaborate, **Model-loss** (Sablayrolles et al., 2019) calculates the mean loss of data point x across all reference models and then subtracts the loss from the target model to eliminate bias induced by data characteristics. In contrast, **Model-calibration** (Watson et al., 2021) uses only reference models that exclude data point x from its training data, allowing it to implement model-based TDD for any new data point. Moreover, **Model-lira** (Carlini et al., 2022a) treats the detection process as a likelihood ratio test, determining whether the rescaled logit value of data point x in the target model originates from models trained on x. Building on `Model-lira`, **Model-fpr** (Ye et al., 2022) designs a detection method that meets the specified arbitrary false positive ratio. **Model-robust** (Zarifzadeh et al., 2024) introduces a robust TDD method that utilizes only one reference model.

A.6.4 QUERY-BASED DECTECTION

The motivation for query-based algorithms stems from two main reasons. Firstly, some of them aim to implement label-only training data detection (TDD), where the target model provides only predicted labels. In such cases, the detector can depend solely on prediction correctness as the detection criterion, which limits the ability to acquire more intricate and effective detection information. To address this limitation, **Query-augment**(`Query-augment`mentation) (Choquette-Choo et al., 2021) proposes obtaining multiple neighbors of a data point x through data augmentation. The correctness of the target model on these augmented data points is then combined to form input features for the auxiliary classifier in the learning-based algorithm. **Query-transfer** (Li & Zhang, 2021), on the other hand, suggests training a surrogate model based on the prediction labels of the target model. The surrogate model is expected to closely resemble the target model and subsequently replace it to provide more detailed prediction results for arbitrary data points, enabling the generation of a more intricate detection criterion. Moreover, **Query-adv**(`Query-adv`ersarial) (Li & Zhang, 2021; Choquette-Choo et al., 2021) considers the distance of a data point from the target model's decision boundary as a detection criterion with the aid of adversarial tools. This is based on the assumption that training data will generally be farther from the decision boundary than test data.

Another type of query-based algorithm does not assume that the target model only returns prediction labels. In these algorithms, additional queries are introduced to provide more information to aid in detection. For example, **`Query-neighbor`** (Jayaraman et al., 2021; Mattern et al., 2023) adds random noise to the data point x and uses the difference between the loss of the target model on x and its average loss on the neighboring points of x as the detection criterion. **`Query-qrm`** (Bertran et al., 2024) collects a large amount of data that is explicitly not from the target model's training data and obtains the scaled logits of the target model on these samples to train a quantile regression model. This quantile regression model can determine the likelihood that x is not part of the target model's training data. Additionally, **`Query-ref`**(`Query-ref`erence) (Wen et al., 2023) makes extra queries for adversarial samples of x generated based on reference models. These samples help to better reflect the differences in the predicted results of x when x is training data versus when it is not.

Remark. `Query-ref` is categorized as a query-based method rather than a reference-based method because of its innovative query sample generation strategy. It is specifically designed to generate suitable query data for image datasets, rather than for tabular or text data.

A.6.5 How to operate a Model-based or Query-based TDD algorithms

In this section, we outline the implementation of Model-based and Query-based algorithms. Specifically, we demonstrate how to train a reference model for focal data x in the Model-based algorithms, as well as how to obtain additional query results in the Query-based algorithms. By following these steps in Alg 1 and Alg 2, you can effectively implement both Model-based and Query-based TDD algorithms.

Algorithm 1: How to train reference models in Model-based TDD algorithms

---

Input: Reference dataset D, focal data x, target model f, number of reference models N;
Output: Whether x was used to train f

for N *times* **do**
  Sample a subset from D ;
  Train a reference model using the combined dataset $D \cup d$;
  Obtain x's detection metric (e.g. loss) from this reference model, which is trained with x ;
  Train a reference model using the dataset $D \setminus d$;
  Obtain x's detection metric (e.g. loss) from this reference model, which is trained without x
end
Obtain x's detection metric from the target model f ;
Implement Model-based TDD using the detection criterion from reference models and target model ;

Algorithm 2: How to obtain extra queries in Query-based TDD algorithms

---

Input: Focal data x, target model $*f*$, number of queries per sample N;
Output: Whether x was used to train f

for N *times* **do**
  Modify the data point x based on the chosen data augmentation strategy (e.g., add noise, flip) ;
  Input the modified data d' into the target model $*f*$ to obtain the query results
end
Implement training data detection using the query results obtained from the different modified data points ;

Table 18: Training accuracy and test accuracy of target models trained on different datasets (corresponds to Table 4 in Section 3.2). WRN28-2, Multilayer Perceptron, and DistilBERT are trained on image, table, and text datasets, respectively. Typically, target models trained on datasets with more categories exhibit smaller test accuracy and greater train-test accuracy gaps.

Modality	Dataset	# Classes	Train accuracy	Test accuracy	Train-test accuracy gap
Image	CIFAR-10	10	0.981	0.877	0.104
	CIFAR-100	100	1.000	0.583	0.417
	BloodMNIST	8	0.989	0.955	0.034
	CelebA	2	0.988	0.976	0.013
Tabular	Purchase	100	1.000	0.897	0.103
	Texas	100	0.766	0.546	0.220
	Adult	2	0.831	0.830	0.001
	Student	3	0.855	0.735	0.121
Text	Rotten tomatoes	2	0.947	0.833	0.113
	Tweet Eval	2	0.840	0.739	0.101
	GLUE-CoLA	2	0.864	0.763	0.100
	ECtHR Articles	13	0.476	0.438	0.038

Table 19: Training accuracy and test accuracy of target models trained with different architectures(corresponds to Table 5 in Section 3.2). CIFAR-10 and Purchase datasets were used to train image models and tabular models from scratch, respectively. The Rotten Tomatoes dataset was used to fine-tune the pre-trained text models.

Dataset	Target model	Train accuracy	Test accuracy	Train-test accuracy gap
CIFAR-10	WRN28-2	0.981	0.877	0.104
	ResNet18	0.992	0.880	0.112
	VGG11	1.000	0.853	0.147
	MobileNet-v2	0.934	0.845	0.089
Purchase	Multilayer Perceptron	1.000	0.897	0.103
	CatBoost	1.000	0.725	0.276
	Logistic Regression	0.999	0.755	0.244
Rotten tomatoes	DistilBERT	0.947	0.833	0.113
	RoBERTa	0.964	0.881	0.083
	Flan-T5	0.911	0.886	0.025

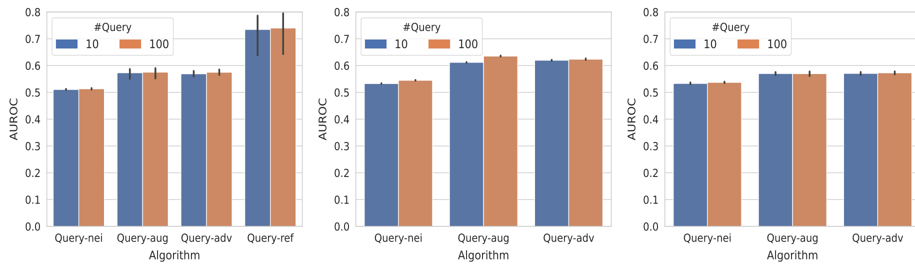
Table 20: Training details for various model architectures, including learning rate, weight decay, maximum training epochs, and more. MLP stands for Multilayer Perceptron, and LR stands for Logistic Regression. 'N/A' indicates that the model does not require consideration of the corresponding hyperparameter.

Modality	Target model	Learning rate	Weight decay	Maximum epochs	Optimizer	Learning rate schedule	Batch size
Image	WRN28-2	0.1	0.0005	200	SGD	Cosine Annealing	256
	ResNet18	0.1	0.0005	200	SGD	Cosine Annealing	256
	VGG11	0.1	0.0005	200	SGD	Cosine Annealing	256
	MobileNet-v2	0.1	0.0005	200	SGD	Cosine Annealing	256
Tabular	MLP	0.001	0.0001	200	Adam	N/A	256
	CatBoost	0.05	N/A	10,000	N/A	N/A	N/A
	LR	N/A	N/A	100	N/A	N/A	N/A
Text	DistilBERT	0.00002	0.01	10	AdamW	N/A	32
	RoBERTa	0.00002	0.01	10	AdamW	N/A	32
	Flan-T5	0.00002	0.01	10	AdamW	N/A	32

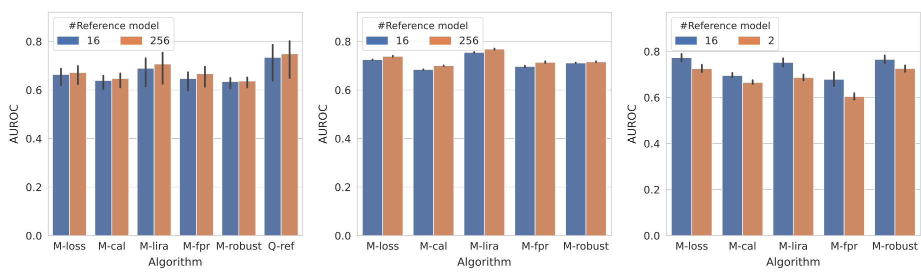
A.7 Performances and training details of target models

A.8 Detection performance with more queries and reference models.

In this part, we analyze the potential of the two latest types of TDD algorithms: reference-based and query-based. Specifically, we expand the number of reference models that the reference-based algorithm can create and the number of queries that the query-based algorithm can handle. Due to limitations in computational resources, we opt to decrease rather than increase the number of reference models in the text modality. We anticipate that similar conclusions can be drawn with a larger number of reference models. The results in Figure 5 demonstrate that increasing the number of queries or reference models can improve the TDD algorithm's performance. Adjusting the number of reference models has a more significant impact on the algorithm's performance compared to increasing the number of queries. Nevertheless, enhancing computational resources does not close the gap in algorithm design. For instance, in tabular data, even with 256 reference models, other algorithms fail to outperform LiRA with 16 reference models.



(a) Image (b) Tabular (c) Text



(d) Image (e) Tabular (f) Text

Figure 5: AUROC of TDD algorithms with more queries and reference models. The abbreviations Query-nei, Q-ref, M-loss, M-cal, M-lira, M-fpr, and M-robust stand for `Query-neighbor`, `Query-ref`, `Model-loss`, `Model-calibration`, `Model-lira`, `Model-fpr`, and `Model-robust`, respectively. While an increase in available computing resources can enhance TDD performance to some extent, the improvement is not significant. Therefore, a more cost-effective approach is to focus on designing more powerful algorithms.

A.9 Complete version of the experimental results

A.10 Performance under different metrics

Table 21: Complete version of TDD performance across different datasets (corresponds to Table 4). WRN28-2, Multilayer Perceptron, and DistilBERT are trained on image, table, and text datasets, respectively.

Modality	Image				Tabular				Text				Avg.
Dataset	CIFAR-10	CIFAR-100	BloodMNIST	CelebA	Purchase	Texas	Adult	Student	Rotten	Tweet	CoLA	ECtHR	
Metric-loss	**0.635** (± **0.053**)	**0.858** (± **0.004**)	**0.527** (± **0.018**)	**0.509** (± **0.012**)	0.619 (± 0.003)	0.629 (± 0.013)	0.500 (± 0.003)	**0.566** (± **0.032**)	**0.582** (± **0.007**)	**0.566** (± **0.005**)	**0.571** (± **0.003**)	0.521 (± 0.007)	0.590 (± 0.094)
Metric-conf	**0.635** (± **0.053**)	**0.858** (± **0.004**)	**0.527** (± **0.018**)	**0.509** (± **0.012**)	0.619 (± 0.003)	0.629 (± 0.013)	0.500 (± 0.003)	**0.566** (± **0.032**)	**0.582** (± **0.007**)	**0.566** (± **0.005**)	**0.571** (± **0.003**)	0.521 (± 0.007)	0.590 (± 0.094)
Metric-corr	0.552 (± 0.009)	0.708 (± 0.002)	0.517 (± 0.006)	0.507 (± 0.002)	0.551 (± 0.001)	0.610 (± 0.012)	**0.501** (± **0.002**)	0.560 (± 0.022)	0.557 (± 0.006)	0.550 (± 0.005)	0.550 (± 0.007)	0.519 (± 0.005)	0.557 (± 0.055)
Metric-ent	0.628 (± 0.058)	0.848 (± 0.004)	0.525 (± 0.018)	0.508 (± 0.010)	0.616 (± 0.003)	0.563 (± 0.010)	0.498 (± 0.003)	0.520 (± 0.018)	0.561 (± 0.007)	0.528 (± 0.005)	0.519 (± 0.003)	0.507 (± 0.016)	0.568 (± 0.096)
Metric-ment	**0.635** (± **0.053**)	**0.858** (± **0.004**)	**0.527** (± **0.018**)	**0.509** (± **0.012**)	**0.620** (± **0.003**)	**0.630** (± **0.013**)	0.500 (± 0.003)	**0.566** (± **0.031**)	**0.582** (± **0.007**)	**0.566** (± **0.005**)	**0.571** (± **0.003**)	**0.522** (± **0.007**)	**0.591** (± **0.094**)
Learn-original	0.631 (± 0.064)	0.870 (± 0.003)	0.508 (± 0.010)	0.503 (± 0.007)	0.652 (± 0.002)	0.597 (± 0.011)	0.502 (± 0.005)	0.531 (± 0.024)	0.558 (± 0.009)	0.529 (± 0.003)	0.568 (± 0.009)	0.506 (± 0.007)	0.580 (± 0.103)
Learn-top3	0.628 (± 0.057)	0.851 (± 0.004)	0.526 (± 0.016)	0.503 (± 0.002)	0.677 (± 0.003)	0.573 (± 0.012)	0.500 (± 0.004)	0.520 (± 0.018)	0.561 (± 0.007)	0.528 (± 0.005)	0.531 (± 0.020)	0.502 (± 0.015)	0.575 (± 0.100)
Learn-sorted	0.628 (± 0.057)	0.850 (± 0.004)	**0.529** (± **0.016**)	**0.508** (± **0.010**)	0.666 (± 0.028)	0.573 (± 0.011)	0.501 (± 0.004)	0.520 (± 0.018)	0.561 (± 0.007)	0.528 (± 0.005)	0.510 (± 0.022)	0.501 (± 0.016)	0.573 (± 0.100)
Learn-label	0.633 (± 0.056)	0.882 (± 0.005)	0.515 (± 0.011)	0.507 (± 0.007)	0.656 (± 0.005)	0.669 (± 0.016)	**0.503** (± **0.003**)	0.590 (± 0.042)	**0.584** (± **0.009**)	**0.570** (± **0.006**)	**0.622** (± **0.010**)	0.517 (± 0.010)	0.604 (± 0.104)
Learn-merge	**0.656** (± **0.065**)	**0.893** (± **0.004**)	0.523 (± 0.017)	0.507 (± 0.001)	**0.684** (± **0.003**)	**0.686** (± **0.017**)	0.502 (± 0.002)	**0.595** (± **0.040**)	**0.584** (± **0.009**)	0.569 (± 0.005)	0.620 (± 0.010)	**0.530** (± **0.004**)	**0.612** (± **0.108**)
Model-loss	0.664 (± 0.050)	0.852 (± 0.004)	**0.560** (± **0.017**)	**0.522** (± **0.004**)	0.725 (± 0.002)	**0.767** (± **0.011**)	**0.509** (± **0.006**)	**0.670** (± **0.068**)	**0.773** (± **0.020**)	**0.756** (± **0.010**)	**0.752** (± **0.017**)	**0.655** (± **0.012**)	**0.684** (± **0.107**)
Model-calibration	0.639 (± 0.040)	0.763 (± 0.005)	0.553 (± 0.016)	0.520 (± 0.004)	0.684 (± 0.002)	0.718 (± 0.011)	0.508 (± 0.006)	0.648 (± 0.063)	0.695 (± 0.012)	0.714 (± 0.006)	0.699 (± 0.014)	0.638 (± 0.011)	0.648 (± 0.082)
Model-lira	**0.690** (± **0.085**)	**0.937** (± **0.002**)	0.536 (± 0.009)	0.512 (± 0.002)	**0.755** (± **0.003**)	0.753 (± 0.007)	0.503 (± 0.002)	0.634 (± 0.063)	0.753 (± 0.024)	0.728 (± 0.007)	0.737 (± 0.014)	0.604 (± 0.014)	0.679 (± 0.126)
Model-fpr	0.647 (± 0.056)	0.852 (± 0.002)	0.552 (± 0.020)	0.516 (± 0.007)	0.697 (± 0.004)	0.723 (± 0.015)	0.507 (± 0.005)	0.641 (± 0.073)	0.679 (± 0.041)	0.722 (± 0.008)	0.708 (± 0.029)	0.635 (± 0.011)	0.657 (± 0.099)
Model-robust	0.635 (± 0.030)	0.889 (± 0.004)	0.552 (± 0.016)	0.520 (± 0.003)	0.711 (± 0.002)	0.762 (± 0.017)	**0.509** (± **0.006**)	0.669 (± 0.061)	0.766 (± 0.022)	0.745 (± 0.008)	0.746 (± 0.014)	0.621 (± 0.013)	0.677 (± 0.112)
Query-augment	0.573 (± 0.025)	0.761 (± 0.010)	0.517 (± 0.008)	0.502 (± 0.002)	0.612 (± 0.001)	**0.612** (± **0.011**)	**0.500** (± **0.002**)	0.560 (± 0.022)	0.570 (± 0.007)	0.551 (± 0.006)	0.561 (± 0.015)	0.518 (± 0.006)	0.570 (± 0.070)
Query-transfer	0.522 (± 0.008)	0.622 (± 0.028)	0.503 (± 0.010)	0.502 (± 0.003)	0.529 (± 0.004)	0.581 (± 0.011)	0.499 (± 0.003)	0.522 (± 0.012)	0.530 (± 0.011)	0.530 (± 0.008)	0.526 (± 0.005)	0.510 (± 0.006)	0.531 (± 0.036)
Query-adv	0.615 (± 0.038)	0.838 (± 0.015)	0.508 (± 0.008)	0.514 (± 0.007)	**0.620** (± **0.003**)	0.579 (± 0.008)	**0.500** (± **0.003**)	**0.563** (± **0.024**)	**0.571** (± **0.007**)	0.551 (± 0.006)	**0.568** (± **0.014**)	0.519 (± 0.005)	0.579 (± 0.089)
Query-neighbor	0.511 (± 0.003)	0.553 (± 0.006)	0.497 (± 0.004)	0.501 (± 0.004)	0.533 (± 0.001)	**0.612** (± **0.007**)	**0.500** (± **0.005**)	0.535 (± 0.015)	0.533 (± 0.004)	**0.556** (± **0.005**)	0.550 (± 0.010)	**0.522** (± **0.014**)	0.534 (± 0.032)
Query-qrm	0.532 (± 0.072)	0.574 (± 0.163)	0.510 (± 0.017)	0.505 (± 0.012)	0.523 (± 0.057)	0.530 (± 0.080)	**0.500** (± **0.003**)	0.526 (± 0.048)	0.524 (± 0.038)	0.521 (± 0.028)	0.511 (± 0.031)	0.512 (± 0.009)	0.522 (± 0.060)
Query-ref	**0.735** (± **0.108**)	**0.941** (± **0.008**)	**0.566** (± **0.022**)	**0.526** (± **0.017**)	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	**0.692** (± **0.176**)

Table 22: Complete version of TDD performance across different target model architectures (corresponds to Table 5). MLP stands for Multilayer Perceptron, and LR stands for Logistic Regression.

Dataset	CIFAR10(Image)				Purchase(Tabular)			Rotten-tomatoes(Text)			Avg.
Target model	WRN28-2	ResNet18	VGG11	MobileNet-v2	MLP	CatBoost	LR	DistilBERT	RoBERTa	Flan-T5	
Metric-loss	**0.635** (± **0.053**)	**0.659** (± **0.042**)	0.684 (± 0.004)	**0.592** (± **0.057**)	0.619 (± 0.003)	0.948 (± 0.009)	0.640 (± 0.003)	**0.582** (± **0.007**)	**0.571** (± **0.019**)	**0.517** (± **0.004**)	**0.645** (± **0.115**)
Metric-conf	**0.635** (± **0.053**)	**0.659** (± **0.042**)	0.684 (± 0.004)	**0.592** (± **0.057**)	0.619 (± 0.003)	0.948 (± 0.009)	0.640 (± 0.003)	**0.582** (± **0.007**)	**0.571** (± **0.019**)	**0.517** (± **0.004**)	**0.645** (± **0.115**)
Metric-corr	0.552 (± 0.009)	0.557 (± 0.006)	0.574 (± 0.003)	0.548 (± 0.019)	0.551 (± 0.001)	0.636 (± 0.001)	0.622 (± 0.002)	0.557 (± 0.006)	0.542 (± 0.014)	0.513 (± 0.002)	0.565 (± 0.036)
Metric-ent	0.628 (± 0.058)	0.654 (± 0.048)	0.680 (± 0.004)	0.582 (± 0.060)	0.616 (± 0.003)	0.943 (± 0.012)	0.594 (± 0.003)	0.561 (± 0.007)	0.555 (± 0.015)	0.509 (± 0.003)	0.632 (± 0.118)
Metric-ment	**0.635** (± **0.053**)	**0.659** (± **0.042**)	**0.685** (± **0.004**)	**0.592** (± **0.056**)	**0.620** (± **0.003**)	**0.950** (± **0.009**)	**0.642** (± **0.003**)	**0.582** (± **0.007**)	**0.571** (± **0.019**)	**0.517** (± **0.004**)	**0.645** (± **0.116**)
Learn-original	0.631 (± 0.064)	0.623 (± 0.031)	0.694 (± 0.004)	0.533 (± 0.017)	0.652 (± 0.002)	0.935 (± 0.092)	0.644 (± 0.008)	0.558 (± 0.009)	0.546 (± 0.015)	0.515 (± 0.007)	0.633 (± 0.121)
Learn-top3	0.628 (± 0.057)	0.653 (± 0.047)	0.680 (± 0.005)	**0.582** (± **0.059**)	0.677 (± 0.003)	0.967 (± 0.019)	0.660 (± 0.008)	0.561 (± 0.007)	0.555 (± 0.015)	0.509 (± 0.003)	0.647 (± 0.124)
Learn-sorted	0.628 (± 0.057)	**0.654** (± **0.048**)	0.680 (± 0.004)	0.578 (± 0.057)	0.666 (± 0.028)	0.963 (± 0.019)	0.661 (± 0.008)	0.561 (± 0.007)	0.555 (± 0.015)	0.509 (± 0.003)	0.646 (± 0.124)
Learn-label	0.633 (± 0.056)	0.612 (± 0.011)	0.707 (± 0.005)	0.557 (± 0.025)	0.656 (± 0.005)	0.954 (± 0.044)	0.701 (± 0.005)	**0.584** (± **0.009**)	0.565 (± 0.020)	**0.520** (± **0.007**)	0.649 (± 0.121)
Learn-merge	**0.656** (± **0.065**)	0.628 (± 0.017)	**0.727** (± **0.006**)	0.528 (± 0.005)	**0.684** (± **0.003**)	**0.968** (± **0.024**)	**0.716** (± **0.006**)	**0.584** (± **0.009**)	**0.566** (± **0.021**)	0.518 (± 0.005)	**0.657** (± **0.128**)
Model-loss	0.664 (± 0.050)	0.709 (± 0.028)	0.729 (± 0.005)	0.607 (± 0.045)	0.725 (± 0.002)	0.975 (± 0.015)	0.776 (± 0.002)	**0.773** (± **0.020**)	**0.656** (± **0.034**)	**0.602** (± **0.017**)	0.721 (± 0.106)
Model-calibration	0.639 (± 0.040)	0.671 (± 0.015)	0.690 (± 0.005)	0.595 (± 0.036)	0.684 (± 0.002)	0.865 (± 0.029)	0.719 (± 0.002)	0.695 (± 0.012)	0.622 (± 0.028)	0.592 (± 0.011)	0.677 (± 0.078)
Model-lira	**0.690** (± **0.085**)	**0.749** (± **0.066**)	**0.780** (± **0.005**)	0.601 (± 0.053)	**0.755** (± **0.003**)	**0.995** (± **0.003**)	0.761 (± 0.005)	0.753 (± 0.024)	0.630 (± 0.028)	0.569 (± 0.009)	**0.728** (± **0.120**)
Model-fpr	0.647 (± 0.056)	0.684 (± 0.033)	0.712 (± 0.006)	**0.619** (± **0.071**)	0.697 (± 0.004)	0.976 (± 0.013)	0.724 (± 0.003)	0.679 (± 0.041)	0.623 (± 0.059)	0.589 (± 0.008)	0.695 (± 0.109)
Model-robust	0.635 (± 0.030)	0.677 (± 0.011)	0.704 (± 0.006)	0.602 (± 0.041)	0.711 (± 0.002)	0.983 (± 0.011)	**0.796** (± **0.003**)	0.766 (± 0.022)	0.639 (± 0.025)	0.574 (± 0.017)	0.709 (± 0.115)
Query-augment	0.573 (± 0.025)	0.575 (± 0.009)	0.633 (± 0.005)	0.542 (± 0.024)	0.612 (± 0.001)	0.696 (± 0.002)	**0.664** (± **0.004**)	0.570 (± 0.007)	0.546 (± 0.020)	0.512 (± 0.009)	0.592 (± 0.057)
Query-transfer	0.522 (± 0.008)	0.522 (± 0.007)	0.533 (± 0.014)	0.507 (± 0.003)	0.529 (± 0.004)	0.587 (± 0.001)	0.574 (± 0.003)	0.530 (± 0.011)	0.515 (± 0.011)	0.506 (± 0.004)	0.533 (± 0.027)
Query-adv	0.615 (± 0.038)	0.621 (± 0.026)	0.666 (± 0.031)	0.583 (± 0.042)	**0.620** (± **0.003**)	0.727 (± 0.002)	0.662 (± 0.005)	**0.571** (± **0.007**)	**0.552** (± **0.020**)	**0.516** (± **0.007**)	0.613 (± 0.063)
Query-neighbor	0.511 (± 0.003)	0.512 (± 0.004)	0.509 (± 0.004)	0.509 (± 0.003)	0.533 (± 0.001)	0.820 (± 0.015)	0.530 (± 0.002)	0.533 (± 0.004)	0.527 (± 0.004)	0.504 (± 0.005)	0.549 (± 0.092)
Query-qrm	0.532 (± 0.072)	0.537 (± 0.083)	0.541 (± 0.088)	0.530 (± 0.060)	0.523 (± 0.057)	**0.946** (± **0.007**)	0.632 (± 0.004)	0.524 (± 0.038)	0.528 (± 0.035)	0.506 (± 0.007)	0.580 (± 0.136)
Query-ref	**0.735** (± **0.108**)	**0.800** (± **0.085**)	**0.843** (± **0.013**)	**0.656** (± **0.103**)	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	**0.759** (± **0.107**)

Table 23: Complete version of TDD performance across various shadow and reference models (corresponds to Table 14). MLP stands for Multilayer Perceptron, and LR stands for Logistic Regression.

Target model	WRN28-2(CIFAR-10)				MLP(Purchase)			DistilBERT(Rotten-tomatoes)			Avg.
Shadow/reference model	WRN28-2	ResNet18	VGG11	MobileNet-v2	MLP	CatBoost	LR	DistilBERT	RoBERTa	Flan-T5	
`Learn-original`	0.631 (± 0.064)	0.578 (± 0.030)	0.632 (± 0.061)	0.539 (± 0.015)	0.652 (± 0.002)	0.651 (± 0.005)	0.564 (± 0.008)	0.558 (± 0.009)	0.560 (± 0.007)	0.546 (± 0.006)	0.591 (± 0.051)
`Learn-top3`	0.628 (± 0.057)	0.628 (± 0.057)	0.628 (± 0.057)	0.628 (± 0.057)	0.677 (± 0.003)	0.646 (± 0.023)	0.515 (± 0.007)	0.561 (± 0.007)	0.561 (± 0.007)	0.561 (± 0.007)	0.603 (± 0.059)
`Learn-sorted`	0.628 (± 0.057)	**0.629** (± **0.058**)	0.628 (± 0.057)	**0.629** (± **0.058**)	0.666 (± 0.028)	**0.656** (± **0.019**)	0.532 (± 0.048)	0.561 (± 0.007)	0.561 (± 0.007)	0.561 (± 0.007)	**0.605** (± **0.058**)
`Learn-label`	0.633 (± 0.056)	0.591 (± 0.024)	0.644 (± 0.060)	0.563 (± 0.014)	0.656 (± 0.005)	0.651 (± 0.005)	0.551 (± 0.009)	**0.584** (± **0.009**)	**0.584** (± **0.009**)	**0.580** (± **0.007**)	0.604 (± 0.045)
`Learn-merge`	**0.656** (± **0.065**)	0.581 (± 0.022)	**0.651** (± **0.063**)	0.509 (± 0.017)	**0.684** (± **0.003**)	0.517 (± 0.019)	**0.595** (± **0.025**)	**0.584** (± **0.009**)	**0.584** (± **0.009**)	**0.580** (± **0.008**)	0.594 (± 0.061)
`Model-loss`	0.664 (± 0.050)	0.657 (± 0.045)	0.641 (± 0.039)	0.632 (± 0.025)	0.725 (± 0.002)	0.608 (± 0.001)	0.611 (± 0.002)	**0.773** (± **0.020**)	0.607 (± 0.008)	0.589 (± 0.008)	0.651 (± 0.061)
`Model-calibration`	0.639 (± 0.040)	0.634 (± 0.033)	0.617 (± 0.032)	0.614 (± 0.019)	0.684 (± 0.002)	0.579 (± 0.001)	0.588 (± 0.002)	0.695 (± 0.012)	0.595 (± 0.007)	0.587 (± 0.007)	0.623 (± 0.043)
`Model-lira`	**0.690** (± **0.085**)	0.659 (± 0.064)	**0.666** (± **0.067**)	0.610 (± 0.025)	**0.755** (± **0.003**)	**0.686** (± **0.003**)	0.588 (± 0.002)	0.753 (± 0.024)	0.602 (± 0.009)	0.553 (± 0.010)	**0.656** (± **0.075**)
`Model-fpr`	0.647 (± 0.056)	**0.668** (± **0.061**)	0.638 (± 0.053)	**0.664** (± **0.056**)	0.697 (± 0.004)	0.645 (± 0.003)	**0.643** (± **0.003**)	0.679 (± 0.041)	0.557 (± 0.007)	0.567 (± 0.008)	0.641 (± 0.055)
`Model-robust`	0.635 (± 0.030)	0.639 (± 0.036)	0.633 (± 0.034)	0.621 (± 0.022)	0.711 (± 0.002)	0.632 (± 0.001)	0.625 (± 0.002)	0.766 (± 0.022)	**0.624** (± **0.010**)	**0.591** (± **0.006**)	0.648 (± 0.053)
`Query-augment`	0.573 (± 0.025)	0.555 (± 0.019)	0.575 (± 0.025)	0.552 (± 0.016)	**0.612** (± **0.001**)	0.612 (± 0.002)	0.612 (± 0.001)	**0.570** (± **0.007**)	**0.569** (± **0.008**)	**0.565** (± **0.011**)	0.580 (± 0.026)
`Query-transfer`	0.522 (± 0.008)	0.529 (± 0.018)	0.518 (± 0.013)	0.518 (± 0.017)	0.529 (± 0.004)	0.535 (± 0.001)	0.529 (± 0.001)	0.530 (± 0.011)	0.529 (± 0.010)	0.514 (± 0.006)	0.525 (± 0.012)
`Query-qrm`	0.532 (± 0.072)	0.532 (± 0.072)	0.533 (± 0.074)	0.532 (± 0.070)	0.523 (± 0.057)	**0.625** (± **0.003**)	**0.622** (± **0.003**)	0.524 (± 0.038)	0.524 (± 0.037)	0.528 (± 0.039)	0.548 (± 0.061)
`Query-ref`	**0.735** (± **0.108**)	**0.740** (± **0.100**)	**0.722** (± **0.093**)	**0.708** (± **0.074**)	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	N/A N/A	**0.726** (± **0.088**)

Table 24: TDD performance across different metrics on WRN28-2 trained on CIFAR-10 dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
`Metric-loss`	**0.579** (± **0.030**)	0.905 (± 0.039)	**0.706** (± **0.033**)	0.623 (± 0.045)	0.095 (± 0.039)	0.659 (± 0.057)	0.246 (± 0.091)	0.009 (± 0.005)	0.132 (± 0.014)	**0.635** (± **0.053**)
`Metric-conf`	**0.579** (± **0.030**)	0.905 (± 0.039)	**0.706** (± **0.033**)	0.623 (± 0.045)	0.095 (± 0.039)	0.659 (± 0.057)	0.246 (± 0.091)	0.009 (± 0.005)	0.131 (± 0.014)	**0.635** (± **0.053**)
`Metric-corr`	0.528 (± 0.004)	**0.982** (± **0.041**)	0.687 (± 0.013)	0.552 (± 0.009)	**0.018** (± **0.041**)	0.877 (± 0.025)	0.105 (± 0.018)	0.000 (± 0.000)	0.000 (± 0.000)	0.552 (± 0.009)
`Metric-ent`	0.574 (± 0.032)	0.878 (± 0.073)	0.694 (± 0.046)	0.614 (± 0.050)	0.122 (± 0.073)	**0.649** (± **0.028**)	0.229 (± 0.101)	**0.012** (± **0.001**)	**0.133** (± **0.015**)	0.628 (± 0.058)
`Metric-ment`	**0.579** (± **0.029**)	0.901 (± 0.044)	0.705 (± 0.035)	**0.624** (± **0.046**)	0.099 (± 0.044)	0.654 (± 0.050)	**0.247** (± **0.091**)	0.010 (± 0.005)	**0.133** (± **0.015**)	**0.635** (± **0.053**)
`Learn-original`	0.570 (± 0.031)	0.872 (± 0.128)	0.688 (± 0.065)	0.611 (± 0.051)	0.128 (± 0.128)	0.650 (± 0.031)	0.222 (± 0.102)	0.015 (± 0.005)	0.162 (± 0.031)	0.631 (± 0.064)
`Learn-top3`	0.576 (± 0.033)	0.877 (± 0.071)	0.695 (± 0.046)	0.616 (± 0.051)	0.123 (± 0.071)	**0.645** (± **0.033**)	0.232 (± 0.102)	0.012 (± 0.001)	0.130 (± 0.013)	0.628 (± 0.057)
`Learn-sorted`	0.575 (± 0.032)	0.875 (± 0.070)	0.694 (± 0.046)	0.615 (± 0.050)	0.125 (± 0.070)	**0.645** (± **0.032**)	0.230 (± 0.101)	0.012 (± 0.001)	0.132 (± 0.014)	0.628 (± 0.057)
`Learn-label`	0.573 (± 0.027)	**0.925** (± **0.061**)	**0.708** (± **0.038**)	0.618 (± 0.045)	**0.075** (± **0.061**)	0.689 (± 0.036)	0.237 (± 0.090)	0.015 (± 0.005)	0.156 (± 0.031)	0.633 (± 0.056)
`Learn-merge`	**0.580** (± **0.030**)	0.908 (± 0.039)	**0.708** (± **0.034**)	**0.625** (± **0.046**)	0.092 (± 0.039)	0.658 (± 0.054)	**0.250** (± **0.093**)	**0.020** (± **0.008**)	**0.178** (± **0.041**)	**0.656** (± **0.065**)
`Model-loss`	0.581 (± 0.026)	0.811 (± 0.053)	0.676 (± 0.022)	0.611 (± 0.032)	0.189 (± 0.053)	0.589 (± 0.086)	0.222 (± 0.065)	0.050 (± 0.015)	0.211 (± 0.027)	0.664 (± 0.050)
`Model-calibration`	0.566 (± 0.020)	0.835 (± 0.040)	0.674 (± 0.018)	0.597 (± 0.027)	0.165 (± 0.040)	0.641 (± 0.069)	0.193 (± 0.054)	0.040 (± 0.011)	0.173 (± 0.013)	0.639 (± 0.040)
`Model-lira`	0.591 (± 0.039)	0.810 (± 0.036)	**0.682** (± **0.026**)	**0.622** (± **0.049**)	0.190 (± 0.036)	0.567 (± 0.108)	**0.243** (± **0.097**)	**0.120** (± **0.059**)	**0.300** (± **0.107**)	**0.690** (± **0.085**)
`Model-fpr`	**0.624** (± **0.042**)	0.520 (± 0.074)	0.566 (± 0.060)	0.605 (± 0.038)	0.480 (± 0.074)	**0.310** (± **0.027**)	0.210 (± 0.076)	0.074 (± 0.034)	0.261 (± 0.064)	0.647 (± 0.056)
`Model-robust`	0.553 (± 0.015)	**0.882** (± **0.135**)	0.676 (± 0.039)	0.583 (± 0.018)	**0.118** (± **0.135**)	0.716 (± 0.123)	0.167 (± 0.036)	0.070 (± 0.023)	0.221 (± 0.032)	0.635 (± 0.030)
`Query-augment`	0.539 (± 0.013)	0.917 (± 0.039)	0.679 (± 0.019)	0.567 (± 0.022)	0.083 (± 0.039)	0.783 (± 0.027)	0.135 (± 0.045)	0.004 (± 0.003)	0.071 (± 0.046)	0.573 (± 0.025)
`Query-transfer`	0.519 (± 0.005)	**0.943** (± **0.033**)	0.670 (± 0.012)	0.535 (± 0.008)	**0.057** (± **0.033**)	0.873 (± 0.019)	0.070 (± 0.017)	0.006 (± 0.003)	0.101 (± 0.003)	0.522 (± 0.008)
`Query-adv`	0.583 (± 0.035)	0.916 (± 0.027)	**0.712** (± **0.031**)	0.631 (± 0.042)	0.084 (± 0.027)	0.654 (± 0.074)	0.261 (± 0.084)	0.002 (± 0.004)	0.084 (± 0.050)	0.615 (± 0.038)
`Query-neighbor`	0.514 (± 0.005)	0.417 (± 0.115)	0.452 (± 0.079)	0.510 (± 0.001)	0.583 (± 0.115)	**0.396** (± **0.116**)	0.021 (± 0.003)	0.000 (± 0.000)	0.074 (± 0.011)	0.511 (± 0.003)
`Query-qrm`	0.518 (± 0.037)	0.583 (± 0.301)	0.522 (± 0.150)	0.529 (± 0.058)	0.417 (± 0.301)	0.525 (± 0.252)	0.057 (± 0.116)	0.000 (± 0.000)	0.000 (± 0.000)	0.532 (± 0.072)
`Query-ref`	**0.649** (± **0.075**)	0.776 (± 0.140)	0.696 (± 0.050)	**0.666** (± **0.062**)	0.224 (± 0.140)	0.445 (± 0.180)	**0.331** (± **0.123**)	**0.152** (± **0.084**)	**0.355** (± **0.152**)	**0.735** (± **0.108**)

Table 25: TDD performance across different metrics on MLP trained on Purchase dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
`Metric-loss`	**0.587** (± **0.002**)	0.956 (± 0.006)	0.727 (± 0.003)	0.642 (± 0.003)	0.044 (± 0.006)	**0.672** (± **0.005**)	0.283 (± 0.007)	**0.010** (± **0.000**)	**0.108** (± **0.001**)	0.619 (± 0.003)
`Metric-conf`	**0.587** (± **0.002**)	0.956 (± 0.006)	0.727 (± 0.003)	0.642 (± 0.003)	0.044 (± 0.006)	**0.672** (± **0.005**)	0.283 (± 0.007)	**0.010** (± **0.000**)	**0.108** (± **0.001**)	0.619 (± 0.003)
`Metric-corr`	0.527 (± 0.002)	**1.000** (± **0.000**)	0.690 (± 0.001)	0.551 (± 0.001)	**0.000** (± **0.000**)	0.897 (± 0.002)	0.103 (± 0.002)	0.000 (± 0.000)	0.000 (± 0.000)	0.551 (± 0.001)
`Metric-ent`	0.582 (± 0.002)	0.948 (± 0.008)	0.721 (± 0.004)	0.634 (± 0.004)	0.052 (± 0.008)	0.680 (± 0.006)	0.268 (± 0.007)	**0.010** (± **0.000**)	**0.108** (± **0.001**)	0.616 (± 0.003)
`Metric-ment`	**0.587** (± **0.002**)	0.962 (± 0.005)	**0.729** (± **0.003**)	**0.643** (± **0.003**)	0.038 (± 0.005)	0.676 (± 0.008)	**0.286** (± **0.007**)	**0.010** (± **0.000**)	**0.108** (± **0.001**)	**0.620** (± **0.003**)
`Learn-original`	0.582 (± 0.003)	0.956 (± 0.010)	0.724 (± 0.002)	0.635 (± 0.002)	0.044 (± 0.010)	0.686 (± 0.011)	0.270 (± 0.005)	0.016 (± 0.002)	0.151 (± 0.005)	0.652 (± 0.002)
`Learn-top3`	0.585 (± 0.002)	0.956 (± 0.006)	0.726 (± 0.003)	0.639 (± 0.003)	0.044 (± 0.006)	0.678 (± 0.005)	0.279 (± 0.007)	0.018 (± 0.001)	0.175 (± 0.010)	0.677 (± 0.003)
`Learn-sorted`	0.586 (± 0.003)	0.949 (± 0.007)	0.725 (± 0.003)	0.640 (± 0.004)	0.051 (± 0.007)	0.670 (± 0.007)	0.280 (± 0.008)	0.018 (± 0.004)	0.167 (± 0.034)	0.666 (± 0.028)
`Learn-label`	0.585 (± 0.002)	**0.965** (± **0.003**)	**0.728** (± **0.002**)	0.640 (± 0.003)	**0.035** (± **0.003**)	0.685 (± 0.006)	0.280 (± 0.006)	0.016 (± 0.000)	0.153 (± 0.004)	0.656 (± 0.005)
`Learn-merge`	**0.589** (± **0.002**)	0.953 (± 0.005)	**0.728** (± **0.002**)	**0.644** (± **0.003**)	0.047 (± 0.005)	**0.665** (± **0.006**)	**0.288** (± **0.006**)	**0.020** (± **0.001**)	**0.187** (± **0.003**)	**0.684** (± **0.003**)
`Model-loss`	0.611 (± 0.006)	0.843 (± 0.028)	0.708 (± 0.006)	0.653 (± 0.001)	0.157 (± 0.028)	0.537 (± 0.030)	0.306 (± 0.003)	0.056 (± 0.002)	0.276 (± 0.003)	0.725 (± 0.002)
`Model-calibration`	0.590 (± 0.011)	0.842 (± 0.063)	0.693 (± 0.016)	0.628 (± 0.002)	0.158 (± 0.063)	0.586 (± 0.065)	0.256 (± 0.003)	0.040 (± 0.002)	0.206 (± 0.002)	0.684 (± 0.002)
`Model-lira`	0.614 (± 0.003)	**0.913** (± **0.022**)	**0.734** (± **0.006**)	**0.670** (± **0.002**)	**0.087** (± **0.022**)	0.573 (± 0.021)	**0.340** (± **0.004**)	**0.134** (± **0.009**)	**0.378** (± **0.005**)	**0.755** (± **0.003**)
`Model-fpr`	**0.636** (± **0.009**)	0.658 (± 0.035)	0.646 (± 0.012)	0.640 (± 0.002)	0.342 (± 0.035)	**0.377** (± **0.032**)	0.281 (± 0.005)	0.073 (± 0.008)	0.296 (± 0.012)	0.697 (± 0.004)
`Model-robust`	0.599 (± 0.012)	0.839 (± 0.074)	0.697 (± 0.018)	0.638 (± 0.002)	0.161 (± 0.074)	0.564 (± 0.074)	0.275 (± 0.005)	0.094 (± 0.003)	0.289 (± 0.003)	0.711 (± 0.002)
`Query-augment`	0.563 (± 0.002)	0.963 (± 0.002)	**0.711** (± **0.002**)	**0.609** (± **0.001**)	0.037 (± 0.002)	0.745 (± 0.003)	**0.218** (± **0.003**)	0.000 (± 0.000)	0.000 (± 0.000)	0.612 (± 0.001)
`Query-transfer`	0.523 (± 0.003)	**0.981** (± **0.008**)	0.682 (± 0.004)	0.544 (± 0.005)	**0.019** (± **0.008**)	0.893 (± 0.002)	0.088 (± 0.009)	**0.010** (± **0.000**)	0.100 (± 0.002)	0.529 (± 0.004)
`Query-adv`	**0.569** (± **0.005**)	0.879 (± 0.038)	0.690 (± 0.008)	0.607 (± 0.003)	0.121 (± 0.038)	0.665 (± 0.038)	0.214 (± 0.006)	0.000 (± 0.000)	0.000 (± 0.000)	**0.620** (± **0.003**)
`Query-neighbor`	0.524 (± 0.006)	0.527 (± 0.094)	0.522 (± 0.046)	0.524 (± 0.001)	0.473 (± 0.094)	0.480 (± 0.093)	0.048 (± 0.002)	0.000 (± 0.000)	**0.115** (± **0.002**)	0.533 (± 0.001)
`Query-qrm`	0.516 (± 0.037)	0.313 (± 0.421)	0.282 (± 0.313)	0.529 (± 0.064)	0.687 (± 0.421)	**0.255** (± **0.313**)	0.058 (± 0.128)	0.000 (± 0.000)	0.000 (± 0.000)	0.523 (± 0.057)

Table 26: TDD performance across different metrics on DistilBERT trained on Rotten-tomatoes dataset. MA(membership advantage) (Jayaraman et al., 2021) equals the difference between the true positive rate and the false positive rate. For all metrics except for FPR and FNR, higher values indicate better performance of the corresponding TDD algorithm.

Algorithm	Precision	Recall	F1-score	Acc	FNR ↓	FPR ↓	MA	TPR@1%FPR	TPR@10%FPR	AUROC
`Metric-loss`	**0.549** (± **0.010**)	0.828 (± 0.031)	0.660 (± 0.013)	0.578 (± 0.009)	0.172 (± 0.031)	0.672 (± 0.017)	0.156 (± 0.019)	**0.011** (± **0.003**)	**0.121** (± **0.010**)	**0.582** (± **0.007**)
`Metric-conf`	**0.549** (± **0.010**)	0.828 (± 0.031)	0.660 (± 0.013)	0.578 (± 0.009)	0.172 (± 0.031)	0.672 (± 0.017)	0.156 (± 0.019)	**0.011** (± **0.003**)	0.121 (± 0.010)	**0.582** (± **0.007**)
`Metric-corr`	0.529 (± 0.009)	**0.947** (± **0.008**)	**0.678** (± **0.008**)	0.557 (± 0.006)	**0.053** (± **0.008**)	0.833 (± 0.006)	0.113 (± 0.011)	0.000 (± 0.000)	0.000 (± 0.000)	0.557 (± 0.006)
`Metric-ent`	0.536 (± 0.009)	0.766 (± 0.061)	0.629 (± 0.020)	0.555 (± 0.007)	0.234 (± 0.061)	**0.655** (± **0.050**)	0.110 (± 0.013)	**0.011** (± **0.003**)	0.119 (± 0.010)	0.561 (± 0.007)
`Metric-ment`	**0.549** (± **0.010**)	0.828 (± 0.031)	0.660 (± 0.013)	**0.578** (± **0.009**)	0.172 (± 0.031)	0.672 (± 0.017)	**0.156** (± **0.019**)	**0.011** (± **0.003**)	**0.121** (± **0.010**)	**0.582** (± **0.007**)
`Learn-original`	0.533 (± 0.010)	0.780 (± 0.053)	0.633 (± 0.017)	0.552 (± 0.009)	0.220 (± 0.053)	0.675 (± 0.050)	0.105 (± 0.017)	0.012 (± 0.003)	0.120 (± 0.007)	0.558 (± 0.009)
`Learn-top3`	0.536 (± 0.009)	0.766 (± 0.061)	0.629 (± 0.020)	0.555 (± 0.007)	0.234 (± 0.061)	**0.655** (± **0.050**)	0.110 (± 0.013)	0.011 (± 0.003)	0.119 (± 0.009)	0.561 (± 0.007)
`Learn-sorted`	0.536 (± 0.009)	0.766 (± 0.061)	0.629 (± 0.020)	0.555 (± 0.007)	0.234 (± 0.061)	**0.655** (± **0.050**)	0.110 (± 0.013)	0.011 (± 0.003)	0.119 (± 0.010)	0.561 (± 0.007)
`Learn-label`	0.546 (± 0.009)	**0.866** (± **0.023**)	**0.670** (± **0.012**)	**0.578** (± **0.009**)	**0.134** (± **0.023**)	0.711 (± 0.019)	0.155 (± 0.018)	0.011 (± 0.003)	0.122 (± 0.007)	0.584 (± 0.009)
`Learn-merge`	**0.547** (± **0.010**)	0.862 (± 0.017)	0.669 (± 0.011)	0.578 (± 0.009)	0.138 (± 0.017)	0.705 (± 0.018)	**0.157** (± **0.019**)	**0.012** (± **0.002**)	**0.122** (± **0.009**)	**0.584** (± **0.009**)
`Model-loss`	**0.683** (± **0.011**)	0.707 (± 0.061)	0.694 (± 0.025)	**0.691** (± **0.015**)	0.293 (± 0.061)	0.324 (± 0.033)	**0.383** (± **0.030**)	0.148 (± 0.021)	**0.385** (± **0.034**)	**0.773** (± **0.020**)
`Model-calibration`	0.606 (± 0.008)	0.777 (± 0.045)	0.680 (± 0.015)	0.639 (± 0.006)	0.223 (± 0.045)	0.500 (± 0.041)	0.277 (± 0.013)	0.106 (± 0.011)	0.234 (± 0.019)	0.695 (± 0.012)
`Model-lira`	0.631 (± 0.007)	**0.813** (± **0.043**)	0.710 (± 0.018)	0.671 (± 0.017)	**0.187** (± **0.043**)	0.471 (± 0.032)	0.342 (± 0.035)	**0.183** (± **0.026**)	0.374 (± 0.048)	0.753 (± 0.024)
`Model-fpr`	0.671 (± 0.006)	0.506 (± 0.094)	0.573 (± 0.062)	0.630 (± 0.025)	0.494 (± 0.094)	**0.245** (± **0.048**)	0.260 (± 0.049)	0.141 (± 0.024)	0.340 (± 0.043)	0.679 (± 0.041)
`Model-robust`	0.651 (± 0.031)	0.797 (± 0.063)	**0.715** (± **0.008**)	0.684 (± 0.021)	0.203 (± 0.063)	0.429 (± 0.101)	0.368 (± 0.042)	0.162 (± 0.028)	0.375 (± 0.027)	0.766 (± 0.022)
`Query-augment`	**0.537** (± **0.008**)	0.878 (± 0.019)	0.666 (± 0.003)	0.565 (± 0.006)	0.122 (± 0.019)	0.747 (± 0.026)	0.131 (± 0.012)	0.000 (± 0.000)	0.000 (± 0.000)	0.570 (± 0.007)
`Query-transfer`	0.525 (± 0.011)	**0.933** (± **0.009**)	**0.672** (± **0.011**)	0.549 (± 0.007)	**0.067** (± **0.009**)	0.835 (± 0.007)	0.098 (± 0.014)	0.008 (± 0.005)	0.090 (± 0.040)	0.530 (± 0.011)
`Query-adv`	0.537 (± 0.008)	0.881 (± 0.031)	0.667 (± 0.010)	**0.566** (± **0.005**)	0.119 (± 0.031)	0.750 (± 0.040)	**0.131** (± **0.011**)	0.000 (± 0.000)	0.000 (± 0.000)	**0.571** (± **0.007**)
`Query-neighbor`	0.525 (± 0.008)	0.909 (± 0.021)	0.665 (± 0.010)	0.548 (± 0.004)	0.091 (± 0.021)	0.813 (± 0.018)	0.096 (± 0.007)	**0.011** (± **0.003**)	**0.091** (± **0.005**)	0.533 (± 0.004)
`Query-qrm`	0.509 (± 0.023)	0.722 (± 0.079)	0.597 (± 0.042)	0.525 (± 0.035)	0.278 (± 0.079)	**0.672** (± **0.035**)	0.049 (± 0.070)	0.000 (± 0.000)	0.000 (± 0.000)	0.524 (± 0.038)