TRACE: Per-Sample Contamination Detection via Transient Training Dynamics

Shashank M. Bharadwaj[*], **Ivan Jaen Marquez**[*], **Vasilis Papageorgiou**[*], **Ashish Priyadarshi**[*]
Department of Computer Sciences
University of Wisconsin–Madison
{sbharadwaj7,ivanjaenm,vpapageorgio,apriyadarsh3}@wisc.edu

Abstract

Benchmark contamination inflates the reported performance of large language models (LLMs) and obscures whether models solve tasks through generalization or memorization. Current detection methods often rely on aggregate statistics, failing to identify specific leaked examples. We propose **TRACE** (**TR**ajectory **A**nalysis for **C**ontamination **E**stimation), a **per-sample** detection framework that leverages *transient training dynamics* obtained from lightweight fine-tuning. Unlike dataset-level approaches, TRACE extracts gradient-, loss-, and representation-based features for individual benchmark examples during controlled, single-sample optimization. Experiments on Mistral-7B-Instruct with MATH, ARC, and MMLU benchmarks show that TRACE's simple linear probes outperform multilayer alternatives, achieving AUROC scores up to 0.80. Our results suggest that transient dynamics provide a scalable signal for detecting contamination at the sample level, showcasing the potential of benchmark decontamination.

1 Introduction

Large language model (LLM) benchmarks serve as the primary measure of progress in the field, guiding research directions, model selection, and deployment decisions. However, these benchmarks are increasingly affected by *data contamination*, the phenomenon where evaluation samples are partially or fully present in the model's pre-training data. This leads to inflated scores and encourages *benchmaxxing*: the over-optimization of models toward benchmark performance rather than genuine.

In this work, we propose **TRACE** (**TR**ajectory **A**nalysis for **C**ontamination **E**stimation), a per-sample contamination detection framework that leverages *transient training dynamics*. Our key insight is that a model's optimization behavior on a given sample differs measurably depending on whether that sample was previously memorized during pre-training. By performing lightweight, single-sample fine-tuning and monitoring how gradients, losses, and representations evolve, TRACE captures distinct features that distinguish contaminated from clean examples.

Contributions. Our main contributions are:

- We introduce TRACE, a novel per-sample contamination detection method that extracts gradient norms, loss trajectories, and representation drift features during transient single-sample optimization.
- We design a controlled artificial contamination setup using Mistral-7B-Instruct fine-tuned on MATH (Hendrycks et al., 2021), ARC (Clark et al., 2018), and MMLU (Hendrycks et al., 2020) benchmarks, enabling rigorous evaluation of detection methods.

[*]Authors listed in alphabetical order.

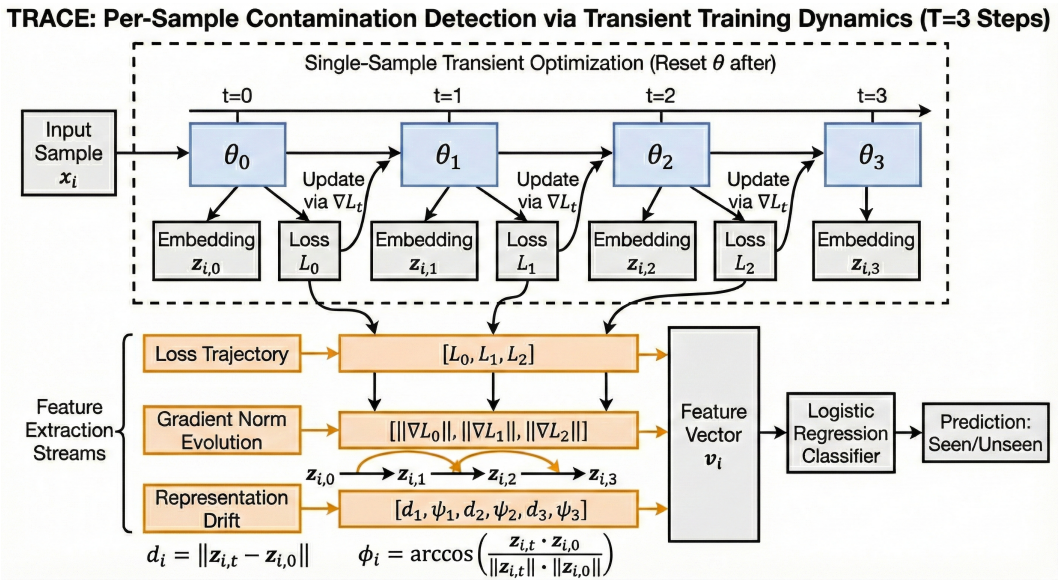

Figure 1: **Overview of TRACE.** For a given input sample x_i, we perform transient single-sample optimization (here shown for $T = 3$ steps). At each step t, we extract the loss L_t, gradient norm $\|\nabla L_t\|$, and embedding $z_{i,t}$. These metrics form a feature vector v_i capturing the sample's training dynamics (loss decay, gradient evolution, and representation drift via d_t and ψ_t), which is then fed into a classifier to detect contamination. The model parameters θ are reset after processing each sample. More details in Section 3. Figure generated using Gemini Nano Banana Pro.

- We demonstrate that TRACE consistently outperforms Min-k% (Shi et al., 2023), a token-level detection baseline, achieving AUC improvements of up to 18.3 percentage points across benchmarks.

2 Related work

Prior work has approached contamination detection and membership inference through various lenses. Xu et al. (2024) estimate leakage by measuring n-gram accuracy and perplexity on paraphrased versions of benchmark datasets. Choi et al. (2025) introduced the **Kernel Divergence Score (KDS)**, which detects contamination by measuring the divergence in embedding kernel matrices before and after fine-tuning on the benchmark samples. Carlini et al. (2022) argue that membership inference attacks must be evaluated at low false-positive rates and introduce the **Likelihood Ratio Attack (LiRA)**, which utilizes Gaussian-fitted shadow model losses to precisely detect if specific examples were used during model training. **Min-k%** (Shi et al., 2023) and its successor **Min-k%++** (Zhang et al., 2024) identify contamination by analyzing the likelihood of tokens with high probabilities, assuming memorized text lies in high-likelihood regions. Zawalski et al. (2025) leverages in-context learning to detect contamination, operating on the principle that providing additional context samples improves model confidence for unseen data but disrupts the memorization patterns of data encountered during training.

In contrast to these works, TRACE leverages per-sample transient training dynamics, specifically the evolution of gradients and representations during single-sample fine-tuning, to identify contamination signals that remain hidden to static inference methods.

3 TRACE: Analysis of Transient Training Dynamics

To detect data contamination, we employ **TRACE**, a sensitivity analysis approach that measures how a model's internal representations shift when exposed to specific data samples during a transient fine-tuning process. A high-level overview can be seen in Figure 1. The core hypothesis is that a model which has effectively memorized a training sample during pre-training will exhibit distinct *transient dynamics*, specifically in gradient norms and representation shifts, compared to a model processing unseen data.

We define a model parametrized by θ. In our setting, θ represents the composition of a frozen base model and a trainable Low-Rank Adaptation (LoRA) adapter. Let $\mathcal{D} = \{x_1, x_2, \ldots, x_M\}$ be a dataset of instruction samples to be evaluated. For each sample $x_i \in \mathcal{D}$, TRACE extracts a *sensitivity trajectory* of embeddings, denoted as $\mathcal{T}_i$, by monitoring the evolution of the sample's latent representation under single-sample optimization.

3.1 Trajectory Extraction Process

For a given sample x_i, the extraction process proceeds as follows:

Initialization. The model is initialized to its reference state θ_0. This state represents the potentially contaminated model being probed.

Baseline Embedding. We compute the initial embedding $z_{i,0}$ of the sample x_i using the parameters θ_0. The embedding is extracted from the last hidden layer of the causal language model. Specifically, to handle variable sequence lengths and padding, we select the hidden state corresponding to the last non-special token in the sequence:

$$z_{i,0} = f(x_i; \theta_0) \tag{1}$$

where $f(\cdot)$ specifically extracts the vector corresponding to the last valid token index k.

Transient Optimization Steps. We perform T steps of Stochastic Gradient Descent (specifically using AdamW) on the typical next-token prediction objective for the single sample x_i. For each step $t \in \{1, \ldots, T\}$:

- Compute the causal language modeling loss $\mathcal{L}_{CLM}(x_i; \theta_{t-1})$.
- Update the trainable parameters (LoRA adapter weights):

$$\theta_t \leftarrow \text{Optim}(\theta_{t-1}, \nabla_\theta \mathcal{L}_{CLM}(x_i; \theta_{t-1})) \tag{2}$$

- Extract the post-update embedding $z_{i,t} = f(x_i; \theta_t)$.

Reset and Independence. Crucially, after computing the full trajectory $\mathcal{T}_i = [z_{i,0}, z_{i,1}, \ldots, z_{i,T}]$ for sample x_i, the model parameters are explicitly **reset** to θ_0. This ensures that TRACE measures the *independent* training dynamics of each sample, unaffected by the optimization history of previous ones.

3.2 Feature Extraction and Contamination Detection

From the raw sensitivity trajectories $\mathcal{T}_i$, we derive a set of scalar features to characterize the optimization dynamics. For a trajectory of length T, we compute the following features for each step t:

Loss Decay Trajectory. The causal language modeling loss values $[\mathcal{L}_{CLM}(x_i; \theta_0), \ldots, \mathcal{L}_{CLM}(x_i; \theta_{T-1})]$ recorded during the transient optimization. We expect rapid drops in loss to signal previously unseen data.

Gradient Norm Evolution. The L2 norm of the gradient vector used for the update at each step, $\|\nabla_\theta \mathcal{L}_{CLM}(x_i; \theta_t)\|_2$.

Representation Drift. We quantify the movement in the representation space relative to the initial state. For each step $t \in \{1, \ldots, T\}$, measuring the shift from θ_0 to θ_t, we compute:

- **L2 Drift**: The Euclidean distance between the current embedding and the initial embedding: $d_t = \|z_{i,t} - z_{i,0}\|_2$.
- **Angular Drift**: The angle between the current embedding vector and the initial embedding vector: $\psi_t = \arccos\left(\frac{z_{i,t} \cdot z_{i,0}}{\|z_{i,t}\| \|z_{i,0}\|}\right)$.

This results in a feature vector $v_i \in \mathbb{R}^{4T}$ for each sample, consisting of T loss values, T gradient norms, T L2 drift values, and T angular drift values.

Contamination Classifier. To detect contamination, we train a binary logistic regression classifier to distinguish between "seen" (contaminated) and "unseen" samples based on the extracted feature vectors v_i. We use class-weighted logistic regression to address potential class imbalance, with standardization applied to normalize features before training.

Note: We deliberately utilize the full optimization trajectory; our preliminary experiments comparing this approach against simple MLP probes trained solely on initial and final states (steps 0 and T) confirmed that leveraging the intermediate dynamics $\{1, \ldots, T-1\}$ is crucial for maximizing detection accuracy.

4 Artificial Contamination Setup

To simulate a controlled contamination environment, we fine-tune a base language model on a mixture of datasets, explicitly marking a subset of samples as "seen" (contaminated). The remaining samples are held out as "unseen" (clean) data for evaluation.

4.1 Data Composition and Contamination

We construct a multi-task dataset combining samples from three diverse benchmarks: MATH (Hendrycks et al., 2021), MMLU (Hendrycks et al., 2020), and ARC (Clark et al., 2018). The datasets are loaded from source files which contain a `seen` label for each sample, which we construct.

Data Mixing The combined dataset aggregates samples from all three sources, ensuring a diverse instruction-following distribution across mathematical reasoning, scientific knowledge, and abstract reasoning tasks.

Contamination Labeling A subset of the training data is flagged as `seen = True`. These samples are included in the fine-tuning set, thereby intentionally contaminating the model with the specific knowledge instantiated in these examples. Samples marked as `seen = False` are excluded from training and instead used during sensitivity analysis to serve as a baseline for unseen data dynamics.

Held-Out Validation Data A separate validation set is maintained, containing entirely unseen samples from all three benchmarks. This validation data is used exclusively for checkpoint selection and evaluation, ensuring unbiased model selection independent of the contaminated samples, and avoiding aggressive overfitting that would make our setup too artificial.

4.2 Model Fine-Tuning

We perform supervised fine-tuning (SFT) using Low-Rank Adaptation (LoRA) to efficiently inject contamination into the model while keeping the base parameters frozen.

Base Model We use `Mistral-7B-Instruct-v0.1` as the base language model for all experiments.

LoRA Configuration LoRA adapters are applied to all linear layers (`all-linear`) with rank $r = 32$ and scaling factor $\alpha = 64$. This relatively high-rank configuration provides sufficient capacity for the model to memorize specific "seen" samples during fine-tuning.

Training and Model Selection The model is trained for up to 10 epochs using a constant learning rate of 5×10^{-5} and an effective batch size of 48. Rather than using the final checkpoint, we periodically evaluate performance on the held-out validation dataset and select the checkpoint with the lowest validation loss. This strategy ensures that the chosen model is well-optimized while retaining generalization capabilities, avoiding excessive overfitting to the contaminated subset.

5 Results

In this section, we present the experimental results of TRACE, using `Mistral-7B-Instruct-v0.1` as the base model, evaluated on three diverse benchmarks: MATH, ARC, and MMLU. As a baseline method, we compare against Min-k% (Shi et al., 2023), a token-level log-probability method for data contamination detection. In our experiments, we use $k = 30\%$ for the Min-k% baseline.

TRACE Trajectory Extraction. For trajectory extraction, we use $T = 5$ optimization steps per sample. Each sample is processed independently: the model is reset to the reference state θ_0 before extracting each trajectory. We use a learning rate of 5×10^{-4} for the transient optimization. The extracted features are then standardized before training the classifier. Finally, we use logistic regression as our classification head.

Train/Evaluation Split. For each benchmark dataset, we maintain separate train and evaluation splits. The TRACE classifier is trained on the train split features and evaluated on the held-out evaluation split. This ensures that our contamination detection performance reflects generalization to unseen samples rather than overfitting to training data.

Compute. For our most compute-heavy experiments, we primarily used a single 80GB A100 through `Google Colab`'s monthly free student credits.

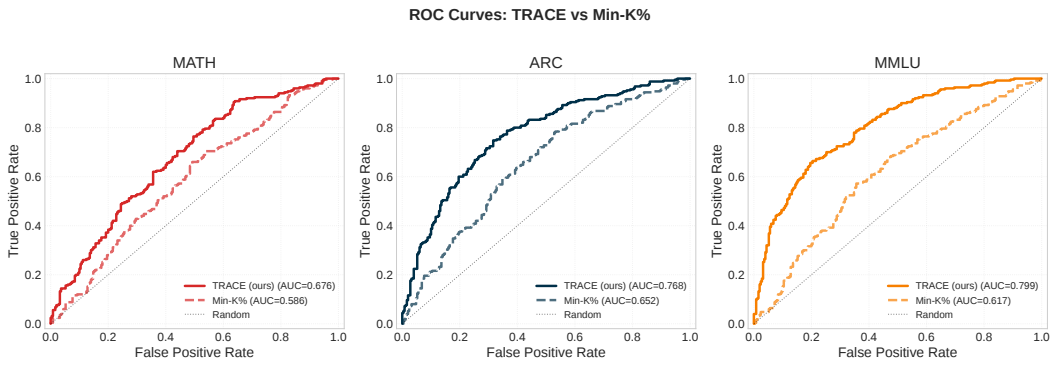

Figure 2: ROC curves, comparing TRACE with Min-k% on all datasets: MATH, ARC, and MMLU. In our setting, TRACE performs better than Min-k%.

5.1 ROC Curves

Figure 2 displays the Receiver Operating Characteristic (ROC) curves comparing TRACE with Min-k% across all three datasets. The ROC curves illustrate the trade-off between the true positive rate (TPR) and false positive rate (FPR) at various classification thresholds.

Across all three benchmarks, TRACE consistently outperforms the Min-k% baseline, with its ROC curves lying significantly above those of Min-k%. This demonstrates that TRACE provides superior discrimination between "seen" (contaminated) and "unseen" samples compared to the token-level probability approach.

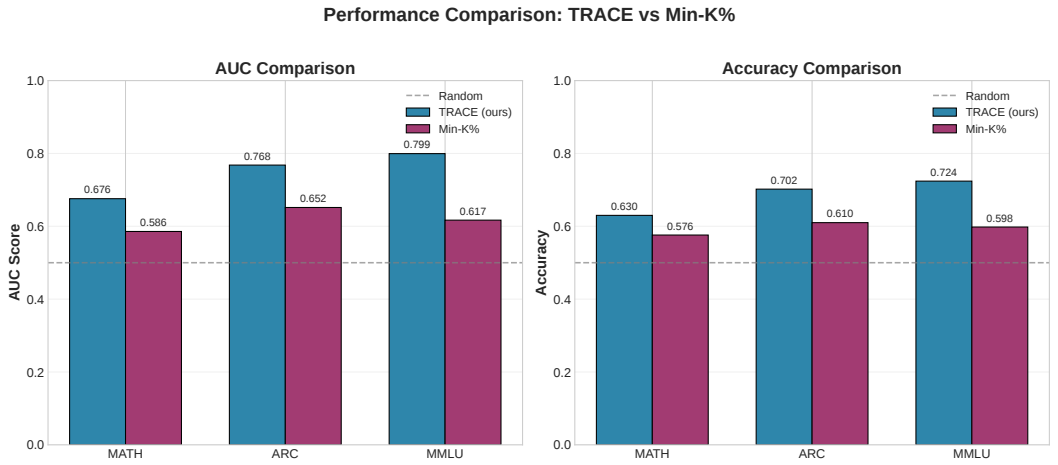

Figure 3: AUC and accuracy, comparing TRACE with Min-k% on all datasets: MATH, ARC, and MMLU. In our setting, TRACE performs better than Min-k%.

5.2 AUC and Accuracy

Figure 3 presents a quantitative comparison of TRACE and Min-k% in terms of Area Under the ROC Curve (AUC) and classification accuracy.

TRACE achieves consistently higher AUC scores across all datasets, with improvements of 9.0 percentage points on MATH (0.676 vs. 0.586), 11.6 percentage points on ARC (0.768 vs. 0.652), and 18.3 percentage points on MMLU (0.800 vs. 0.617). Similarly, TRACE demonstrates higher classification accuracy, with gains of 5.4 percentage points on MATH (63.0% vs. 57.6%), 9.2 percentage points on ARC (70.2% vs. 61.0%), and 12.6 percentage points on MMLU (72.4% vs. 59.8%).

The largest performance gap is observed on MMLU, where TRACE achieves an AUC of 0.800, approaching practical utility for real-world contamination detection scenarios. This superior performance can be attributed to TRACE's ability to capture richer signals through gradient norms, loss trajectories, and representation dynamics during transient optimization, unlike Min-k%, which relies solely on static token probabilities.

5.3 Logistic Regression Feature Importance

Figure 4 presents the feature importance coefficients from the logistic regression classifier used in TRACE across the three datasets. These coefficients reveal which features are most predictive of contamination and how their importance varies by dataset.

A notable observation is that the relative importance of different feature types varies noticably across datasets. This dataset-dependent variation in feature importance is itself an interesting finding, suggesting that different types of benchmark tasks leave distinct features in the model's transient training dynamics. The fact that TRACE can leverage

Figure 4: Feature importance of the logistic regression classifier of TRACE in MATH, ARC, and MMLU.

dataset-specific feature patterns while maintaining strong overall performance across all benchmarks highlights the flexibility and robustness of our approach.

5.4 Sensitivity Analysis of Min-k%

We additionally evaluated the Min-k% baseline across different values of k to ensure a fair comparison. Table 1 reports the AUC and accuracy for $k \in \{10, 20, 30, 50\}$%.

Table 1: Min-k% performance across different values of k.

Dataset	$k = 10\%$	$k = 20\%$	$k = 30\%$	$k = 50\%$
MATH	0.521	0.573	0.586	0.596
ARC	0.656	0.664	0.652	0.639
MMLU	0.623	0.625	0.617	0.607

The results show that while the optimal k value varies by dataset, even the best-performing Min-k% configuration for each dataset still underperforms TRACE. For MATH, higher k values perform better (0.596 at $k = 50\%$), while for ARC and MMLU, lower k values are preferable (0.664 at $k = 20\%$ for ARC; 0.625 at $k = 20\%$ for MMLU). Notably, TRACE's performance advantage persists regardless of the choice of k, confirming that transient training dynamics provide fundamentally different, and more informative, contamination signals than token-level probabilities.

6 Limitations

Simulated Contamination. Our experiments rely on artificially injected contamination, where we explicitly fine-tune the model on hand labeled samples. While this controlled setup enables smooth evaluations, it may not fully capture the complexity of real-world contamination scenarios. In practice, contamination occurs during pre-training on massive web corpora, potentially involving partial matches, paraphrased content, or indirect exposure through data augmentation. Future work should validate TRACE on models with naturally occurring contamination.

Computational Overhead. TRACE requires performing transient fine-tuning for each sample being evaluated, which introduces computational costs compared to purely inference-

based methods like Min-k%. While LoRA-based adaptation mitigates this overhead, scaling TRACE to very large evaluation sets or resource-constrained settings remains a practical consideration.

Limited Baseline Comparison. We compare TRACE primarily against Min-k%, a single representative baseline from the family of token-probability methods. A more comprehensive evaluation would include other detection approaches, providing a clearer picture of TRACE's relative strengths and weaknesses.

Single Model Evaluation. Our experiments are conducted exclusively on Mistral-7B-Instruct. The effectiveness of TRACE may vary across different model architectures, sizes, and training regimes. Evaluating on a broader range of models is essential to establish the generalizability of our approach.

7 Conclusion

We introduced TRACE, a per-sample contamination detection framework that leverages transient training dynamics to identify benchmark leakage in LLMs. By performing lightweight single-sample fine-tuning and extracting gradient norms, loss trajectories, and representation drift features, TRACE captures optimization features that distinguish contaminated from clean examples. Experiments on Mistral-7B-Instruct across MATH, ARC, and MMLU benchmarks demonstrate that TRACE consistently outperforms Min-k%, achieving AUC improvements of up to 18.3 percentage points. Our results suggest that transient dynamics provide a strong signal for sample-level contamination detection, offering a promising direction toward more reliable benchmark evaluation of LLMs.

References

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, 2022.

Hyeong Kyu Choi, Maxim Khanov, Hongxin Wei, and Yixuan Li. How contaminated is your benchmark? quantifying dataset leakage in large language models with kernel divergence. *arXiv preprint arXiv:2502.00678*, 2025.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 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. *arXiv preprint arXiv:2310.16789*, 2023.

Ruijie Xu, Zengzhi Wang, Run-Ze Fan, and Pengfei Liu. Benchmarking benchmark leakage in large language models. *arXiv preprint arXiv:2404.18824*, 2024.

Michał Zawalski, Meriem Boubdir, Klaudia Bałazy, Besmira Nushi, and Pablo Ribalta. Detecting data contamination in llms via in-context learning. *arXiv preprint arXiv:2510.27055*, 2025.

Jingyang Zhang, Jingwei Sun, Eric Yeats, Yang Ouyang, Martin Kuo, Jianyi Zhang, Hao Frank Yang, and Hai Li. Min-k%++: Improved baseline for detecting pre-training data from large language models. *arXiv preprint arXiv:2404.02936*, 2024.