Efficient Ensembling Improves Training Data Attribution

Anonymous authors
Paper under double-blind review

Abstract

Training data attribution (TDA) methods aim to quantify the influence of individual training data points on the model predictions, with broad applications in data-centric AI, such as mislabel detection, data selection, and copyright compensation. However, existing methods in this field, which can be categorized as *retraining-based* and *gradient-based*, have struggled with the trade-off between computational efficiency and attribution efficacy. Retraining-based methods can accurately attribute complex non-convex models but are computationally prohibitive, while gradient-based methods are efficient but often fail for non-convex models. Recent research has shown that augmenting gradient-based methods with ensembles of multiple independently trained models can achieve significantly better attribution efficacy. However, this approach remains impractical for very large-scale applications.

In this work, we discover that expensive, fully independent training is unnecessary for ensembling the gradient-based methods, and we propose two efficient ensemble strategies, Dropout Ensemble and LoRA Ensemble, alternative to naive independent ensemble. These strategies significantly reduce training time (up to 80%), serving time (up to 60%), and space cost (up to 80%) while maintaining similar attribution efficacy to the naive independent ensemble. Our extensive experimental results demonstrate that the proposed strategies are effective across multiple TDA methods on diverse datasets and models, including various generative settings, significantly advancing the Pareto frontier of TDA methods with better computational efficiency and attribution efficacy. We conduct a theoretical analysis that provides insights into the success of our empirical findings.

1 Introduction

Training data plays an increasingly crucial role in modern artificial intelligence (AI) models (Kaplan et al., 2020). Consequently, data-centric AI emerges as a vital paradigm, emphasizing the collection, curation, and understanding of training data. *Training Data Attribution* (TDA) is a family of methods that assess the influence of each training sample on a model's output. Numerous TDA methods have been developed and applied to a wide range of data-centric AI applications, such as mislabel detection (Koh & Liang, 2017), data selection (Engstrom et al., 2024), and copyright compensation (Deng & Ma, 2023), thereby gaining increasing popularity.

However, accurately attributing the training data influence on very large-scale AI applications remains an open challenge. Existing TDA methods can be generally categorized into two groups: *retraining-based methods* and *gradient-based methods* (Hammoudeh & Lowd, 2024). Retraining-based methods involve the systematic retraining of the model with and without specific training samples to observe changes in the model's output (Ghorbani & Zou, 2019; Jia et al., 2019; Feldman & Zhang, 2020; Ilyas et al., 2022; Wang & Jia, 2023). Such methods often require thousands of model retraining, sometimes even growing with the size of the training data, to achieve satisfactory performance, which makes them infeasible for moderately large models. Gradient-based methods, on the other hand, estimate the influence by tracking the gradients of data samples (Koh & Liang, 2017; Yeh et al., 2018; Pruthi et al., 2020). These methods are typically computationally more efficient as they do not require the training of multiple models. But empirically they can be brittle in handling complex non-convex models (Basu et al., 2020; Bae et al., 2022) and be sensitive to the randomness associated with model initialization and training dynamics (Søgaard et al., 2021).

Recent studies have shown that gradient-based TDA methods can be significantly improved by ensembling tens of models independently trained with different random seeds, leading to state-of-the-art attribution efficacy on modern neural network models (Søgaard et al., 2021; Park et al., 2023). Aggregating these independently trained models helps to mitigate the randomness introduced by training dynamics, such as random initializations and stochastic optimization. However, despite its impressive performance, scaling such a naive independent ensemble approach further for very large models remains challenging due to the significant computational costs associated with training multiple models.

In this work, we hypothesize that training models independently is unnecessary for the purpose of TDA ensemble, and we propose two efficient ensemble strategies alternative to the naive independent ensemble approach. Our first strategy, Dropout Ensemble, is motivated by dropout (Srivastava et al., 2014), a common deep learning module initially designed to efficiently approximate ensemble. Dropout Ensemble reduces the computational costs by replacing the independently trained models in the naive ensemble approach with multiple dropout-masked models using the same original model. Our second strategy, LoRA Ensemble, is motivated by an efficient fine-tuning technique, LoRA (Hu et al., 2021). Similar to Dropout Ensemble, LoRA Ensemble reduces the computational costs by replacing the independently trained models with LoRA fine-tuned models from the same original model, which is particularly suitable for generative Transformer models (Vaswani et al., 2017).

We evaluate the proposed Dropout Ensemble and LoRA Ensemble with extensive experiments. Our experiments span various datasets (MNIST (LeCun et al., 1998), CIFAR (Krizhevsky et al., 2009), and MAESTRO (Hawthorne et al., 2018)), model architectures (Multi-Layer Perceptrons (MLP), Residual Neural Networks (ResNet) (He et al., 2016), and Transformers (Vaswani et al., 2017)), and TDA methods (TRAK (Park et al., 2023), Influence Function (Koh & Liang, 2017), and Grad-Dot/Grad-Cos (Charpiat et al., 2019)). Compared to the naive independent ensemble approach, Dropout Ensemble and LoRA Ensemble can significantly reduce the training time, serving time, and space costs by respectively up to 80%, 60%, and 80% while maintaining similar TDA efficacy.

Additionally, we present a theoretical analysis to demonstrate the effectiveness of our proposed efficient ensemble strategies over the naive independent ensembling method. Specifically, we first introduce a general ensemble scheme: *two-step ensemble estimator*, which includes both Dropout Ensemble and LoRA Ensemble. *Two-step ensemble estimator* utilizes a number of base estimators and generates several variants on top of each of these base estimators. Under mild assumptions, our theory shows that the *two-step ensemble estimator* achieves a smaller squared error than the naive ensemble estimator when estimating the optimal attribution score obtained from any given TDA method, given the same number of independently trained checkpoints. Moreover, the gap in squared error between the two estimators remains small when the respective numbers of independently trained checkpoints are large under the same number of ensembles.

We summarize the contributions of this work as follows:

- We show that fully independent training is not a strict requirement for effective TDA with ensembles, which opens up a promising direction for developing more efficient and effective TDA methods.
- We provide theoretical analysis that compares the TDA method with naive independent ensemble and the two novel efficient ensemble strategies: Dropout Ensemble and LoRA Ensemble.
- Our proposed Dropout Ensemble and LoRA Ensemble achieve significant efficiency improvement across a diverse range of machine learning models, datasets, and TDA methods, advancing the state-of-the-art of TDA.

2 Related Work

TDA methods quantify the influence of each training sample on the model predictions by assigning a TDA score to the sample. These methods can be categorized into retraining-based ones and gradient-based ones (Hammoudeh & Lowd, 2024), and our work focuses on the latter. Please refer to Appendix A for more detailed related works.

Ensembling for gradient-based TDA methods. Recent studies have shown the effectiveness of ensembling for improving TDA scores computed with gradient-based methods (Søgaard et al., 2021; Park

et al., 2023), which mitigates their typical issues associated with non-convexity and sensitivity to randomness. Ensembling normally applies the TDA method to many independently trained models. Either averaging the final TDA scores (Søgaard et al., 2021) or aggregating some intermediate terms for score calculation (Park et al., 2023). Besides independently trained models, Park et al. (2023) suggest that model checkpoints at different stages of a single training can be used for ensembling as well. All these ensembling methods, though effective, also require a non-trivial amount of ensembles to perform well. Empirical studies show that their TDA performance suffers significantly when ensemble size is limited (Park et al., 2023). Consequently, the ensemble size, and thus the cost associated with each ensemble, poses a significant barrier to the effective use of ensembling in current TDA methods.

3 Method

Following our hypothesis that independently trained models are unnecessary for ensembling TDA methods, we propose efficient ensemble strategies alternative to the naive independent ensemble.

3.1 Preliminaries

We start by formalizing the TDA problem and the naive independent ensemble method for TDA.

The TDA problem. We have a training set $\mathcal{S} = \{x_1, \ldots, x_n\}$ where each $x_i \in \mathcal{X}_{\text{train}}$, a test set $\mathcal{T} = \{x_1, \ldots, x_m\}$ where each $x_i \in \mathcal{X}_{\text{test}}$, and a trained model output function f_Θ that is parameterized by Θ. Here $\mathcal{X}_{\text{train}}$ and $\mathcal{X}_{\text{test}}$ are the space of the training and test data respectively. We consider both supervised learning and generative modeling settings. For the supervised learning setting, $\mathcal{X}_{\text{train}}$ and $\mathcal{X}_{\text{test}}$ are typically identical, and each element of them corresponds to a data point with both the feature and label. For the generative modeling setting, $\mathcal{X}_{\text{train}}$ refers to the space of training data (e.g., text sequence segments for autoregressive language models) while $\mathcal{X}_{\text{test}}$ refers to the space of model generation. For the model output function f_Θ, typically we will use the model learned from the training set $\mathcal{S}$ (i.e., the model parameters will be chosen as $\Theta^* = \text{argmin}_\Theta \sum_{x_i \in \mathcal{S}} \mathcal{L}(x_i; f_\Theta)$ for some loss function $\mathcal{L}$). However, this is not always the case in practice (Pruthi et al., 2020). For a training set $\mathcal{S}$ and any test sample $x \in \mathcal{T}$, a TDA method τ derives the TDA scores $\tau(x, \mathcal{S}; f_\Theta) \in \mathbb{R}^n$ to quantify the influence of each training data point in $\mathcal{S}$ on the model learned from $\mathcal{S}$. More specifically, the i-th element, $\tau(x, \mathcal{S}; f_\Theta)_i \in \mathbb{R}$, is a real-valued score indicating the importance of x_i on the model output on the test sample x. In the supervised classification setting, the "model output on x" usually refers to the loss, (log-)likelihood, or logit for the model predicting the correct class of x (Koh & Liang, 2017; Park et al., 2023); in the generative setting, it typically refers to the (log-)likelihood for the model generating x (Deng & Ma, 2023).

The naive independent ensemble method for TDA. To mitigate the randomness led by the training dynamics of non-convex deep learning models, the naive independent ensemble method first trains a set of I independent models, $\{\Theta^{(i)}\}_{i=1}^I$, and then derives ensembled TDA scores $\tau_{\text{ens}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^I) \in \mathbb{R}^n$ based on the set of models.

The ensembled TDA scores could be simply obtained by averaging over the TDA scores for individual models, i.e.,

$$\tau_{\text{ens}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^I) = \frac{1}{I} \sum_{i=1}^{I} \tau(x, \mathcal{S}; f_{\Theta^{(i)}}). \tag{1}$$

They could also come from more sophisticated aggregation over the individual models. For example, the TRAK method (Park et al., 2023) works as the following:

$$\tau_{\text{TRAK}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^I) = \left(\frac{1}{I} \sum_{i=1}^{I} \mathbf{Q}_{f_{\Theta^{(i)}}} \right) \cdot \left(\frac{1}{I} \sum_{i=1}^{I} \phi_{f_{\Theta^{(i)}}} \left(\Phi_{f_{\Theta^{(i)}}}^\top \Phi_{f_{\Theta^{(i)}}} \right)^{-1} \Phi_{f_{\Theta^{(i)}}}^\top \right), \tag{2}$$

where $\Theta^{(i)}$ are parameters of models independently trained on the training data; $\mathbf{Q}_{f_{\Theta^{(i)}}}$ is a diagonal matrix with each diagonal element corresponding to the "one minus correct-class probability" of a training data

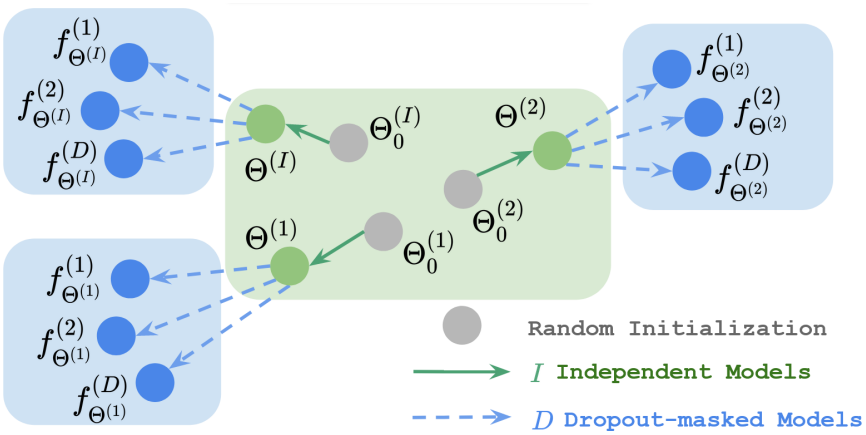


Figure 1: Dropout Ensemble consists of two steps: (1) train I models ($\{\Theta^{(i)}\}_{i=1}^{I}$) independently; (2) get D dropout-masked models ($\{f_{\Theta^{(i)}}^{(d)}\}_{d=1}^{D}$) for each $i = 1, \ldots, I$.

point under model $\Theta^{(i)}$; $\phi_{f_{\Theta^{(i)}}}$ is the vector gradient of $f_{\Theta^{(i)}}(x)$ with respect to $\Theta^{(i)}$; and $\Phi_{f_{\Theta^{(i)}}}$ is the matrix of the vector gradients of $f_{\Theta^{(i)}}(x_j)$ stacked over the training samples $x_j \in S$.[1]

3.2 Computational costs of TDA methods

While recent research has shown that ensembling gradient-based TDA methods can achieve decent attribution efficacy with tens of independently trained models (as opposed to hundreds or even thousands of model retraining in retraining-based TDA methods) (Park et al., 2023), it is still impractical for very large-scale or edge-device applications due to the increased time and space costs. To better quantify the time and space costs associated with TDA methods, we categorize the costs into three parts: training time cost, serving time cost, and storage cost. We provide a detailed explanation for these costs as follows:

- **Training time cost**: This refers to the computational time incurred by processing and training models to be used by TDA ensembles. This is the major computational challenge for ensembling TDA methods.
- **Serving time cost**: This refers to the remaining computational time to deploy TDA ensembles *after model training*. Typically, ensembling gradient-based TDA methods derives different TDA scores using multiple trained models and then aggregates them. The serving time cost refers to the total computational costs incurred during this process, i.e., calculating the TDA scores τ_{ens} for all test data $x_i \in \mathcal{T}$.
- **Space cost**: This refers to the parameters of the trained models that are stored by TDA ensembles. These model parameters are required to run forward and backward passes on top of the models for TDA score calculation. The space costs are generally proportional to the model size.

In this paper, we mainly focus on use cases where a fixed test set exists, such as data selection (Xia et al., 2024). To further discuss this topic, we include a more detailed discussion in Appendix D.1.

3.3 Dropout Ensemble

General methodology. Dropout, a common module used in many modern deep learning models, is initially designed as an efficient approximation of ensemble (Srivastava et al., 2014). Motivated by this idea, we propose a simple, efficient ensemble strategy, Dropout Ensemble, for TDA methods. Instead of performing the TDA ensemble over a large number of independently trained models, the proposed method utilizes multiple dropout masks on the same model to perform the TDA ensemble.

In practice, Dropout Ensemble consists of two steps as illustrated in Figure 1. In the first step, we train I independent models, $\{\Theta^{(i)}\}_{i=1}^{I}$, as shown by the green arrows in Figure 1. Next, as shown by the blue arrows in Figure 1, for each model $\Theta^{(i)}, i \in \{1, \ldots I\}$, we obtain D variants of the model with different dropout masks,

[1]Some details of the exact TRAK implementation are omitted here.

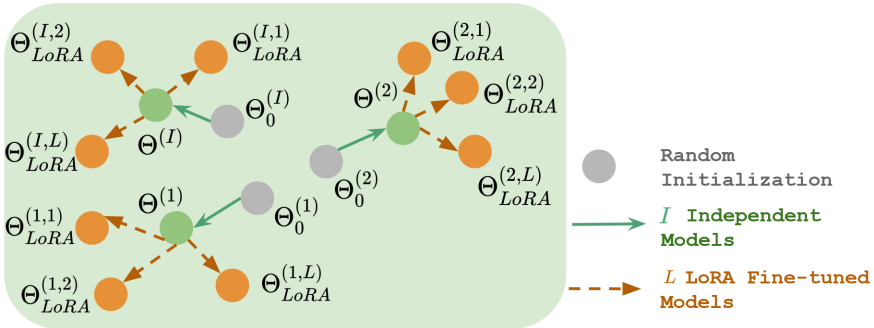


Figure 2: LoRA Ensemble consists of two steps: (1) train I models ($\{\Theta^{(i)}\}_{i=1}^{I}$) independently; (2) get L LoRA fine-tuned models ($\{\Theta^{(i,l)}_{\text{LoRA}}\}_{l=1}^{L}$) for each $i = 1, \ldots, I$.

which are denoted as $\{f^{(d)}_{\Theta^{(i)}}\}_{d=1}^{D}$. For any TDA method, Dropout Ensemble calculates the ensembled TDA scores through $\tau_{\text{ens}}(x, \mathcal{S}; \{f^{(d)}_{\Theta^{(i)}}\}_{1\le i\le I, 1\le d\le D})$.

In comparison to the naive independent ensemble method, Dropout Ensemble can **significantly reduce the training time and space costs**. As we will show in Section 4.2, this method allows us to replace the expensive independently trained models with dropout-masked models that incur no additional training cost or model parameters. There is some serving time cost overhead caused by Dropout Ensemble . As we will show in Appendix D.1, the overhead is small and Dropout Ensemble could achieve better efficacy-efficiency trade-off.

TDA-method-specific optimization. It is possible to further optimize the proposed ensemble strategy when applying it to a particular TDA method. For example, we develop a variant of Dropout Ensemble tailored for the TRAK method to reduce the serving time cost of Dropout Ensemble. Recall that in Eq. (2), the quantities Q's only involve the forward pass of the models on the training data, while ϕ's and Φ's require the backward pass on the training data. We find that, perhaps a bit surprisingly, if we only use the dropout-masked models to calculate Q's while using the original models to calculate ϕ's and Φ's, we can get similar attribution efficacy. Concretely, directly applying Dropout Ensemble on TRAK leads to

$$\tau_{\text{ens}}\left(x, \mathcal{S}; \{f^{(d)}_{\Theta^{(i)}}\}_{\substack{1\le i\le I\\ 1\le d\le D}}\right) = \left(\tfrac{1}{I\cdot D}\textstyle\sum_{i=1}^{I}\sum_{d=1}^{D}\mathbf{Q}_{f^{(d)}_{\Theta^{(i)}}}\right)\cdot\left(\frac{1}{I\cdot D}\sum_{i=1}^{I}\sum_{d=1}^{D}\phi_{f^{(d)}_{\Theta^{(i)}}}\left(\Phi^{\top}_{f^{(d)}_{\Theta^{(i)}}}\Phi_{f^{(d)}_{\Theta^{(i)}}}\right)^{-1}\Phi^{\top}_{f^{(d)}_{\Theta^{(i)}}}\right),$$

while in the optimized version, we have

$$\tau_{\text{ens}}\left(x, \mathcal{S}; \{f^{(d)}_{\Theta^{(i)}}\}_{\substack{1\le i\le I\\ 1\le d\le D}}, \{f_{\Theta^{(i)}}\}_{1\le i\le I}\right) = \left(\tfrac{1}{I\cdot D}\textstyle\sum_{i=1}^{I}\sum_{d=1}^{D}\mathbf{Q}_{f^{(d)}_{\Theta^{(i)}}}\right)\cdot\left(\frac{1}{I}\sum_{i=1}^{I}\phi_{f_{\Theta^{(i)}}}\left(\Phi^{\top}_{f_{\Theta^{(i)}}}\Phi_{f_{\Theta^{(i)}}}\right)^{-1}\Phi^{\top}_{f_{\Theta^{(i)}}}\right).$$

In this case, we only need to evaluate the forward pass on the dropout-masked models and reduce the serving time cost by avoiding the backward pass. We call this variant of our method as Dropout Ensemble (forward-only).

3.4 LoRA Ensemble

For large-scale generative models, especially Transformer models (Vaswani et al., 2017), LoRA adapters have shown great success for efficiently fine-tuning the models (Hu et al., 2021). Our second efficient ensemble strategy, LoRA Ensemble, is motivated by the LoRA techniques. In particular, we propose to replace the independently trained models in the naive ensemble method with LoRA fine-tuned models.

Similar to Dropout Ensemble, LoRA Ensemble also consists of two steps as illustrated in Figure 2. The first step trains I independent models, $\{\Theta^{(i)}\}_{i=1}^{I}$ (green arrows in Figure 2), while the second step further trains L LoRA fine-tuned models, $\{\Theta^{(i,l)}_{\text{LoRA}}\}_{l=1}^{L}$, for each $\Theta^{(i)}$, $i \in \{1, \ldots, I\}$ (brown arrows in Figure 2).

In comparison to Dropout Ensemble, LoRA Ensemble will introduce a small amount of additional training time cost and model parameters for LoRA fine-tuning. However, since we only use the parameters in LoRA adapters to compute the TDA scores, LoRA Ensemble can reduce the serving time cost compared to either dropout-masked models in Dropout Ensemble or independent models in the naive method. This leads to a unique advantage for LoRA Ensemble if the serving time cost is critical among the computational costs.

4 Experiments

4.1 Experimental setup

We conduct extensive experiments on a wide range of settings.

TDA methods. We apply the proposed methods to various gradient-based TDA methods, including influence function (IF) (Koh & Liang, 2017), Grad-Dot/Grad-Cos (Charpiat et al., 2019), and TRAK (Park et al., 2023). The detailed descriptions and implementations of these algorithms are provided in Appendix B.1.

Datasets and models. We consider models with different architectures trained on diverse datasets: (1) a three-layer MLP classifier trained on the MNIST-10 dataset (LeCun et al., 1998), (2) a three-layer MLP classifer and a ResNet-9 classifier (He et al., 2016) trained on the CIFAR-2 dataset (a two-class subset of the CIFAR-10 dataset (Krizhevsky et al., 2009)), and (3) a Music Transformer (Huang et al., 2018) trained on the MAESTRO dataset (Hawthorne et al., 2018). Notably, the former two are supervised classification settings, while the last one is a generative modeling setting. We sample 5000 training samples and 500 test samples from MNIST-10 and CIFAR-2 datasets. For MAESTRO dataset, we sample 5000 training samples and 178 generated samples. More detailed setups for each dataset and model are listed in Appendix C.1. MNIST-10 dataset holds CC BY-SA 3.0 license. CIFAR-10 dataset holds CC-BY 4.0 license. MAESTRO dataset holds CC BY-NC-SA 4.0 license. We also include larger experiment settings in Section 4.4.

Evaluation metric for TDA efficacy. We utilize the linear datamodeling score (LDS) (Park et al., 2023) to evaluate the efficacy of the TDA methods augmented by different ensembling methods. Intuitively, LDS measures the rank correlation between the TDA scores among training samples and the change of model outputs by removing a subset of training samples and retraining the model from scratch. A higher LDS value corresponds to a better alignment between the TDA scores and the influence of the training samples on the model outputs, thus better TDA quality. We refer the reader to Appendix B.2 for the exact definition of LDS.

Evaluation metrics for TDA efficiency. As introduced in Section 3.2, we measure the TDA efficiency in terms of the training time cost, serving time cost, and space cost. For both training and serving time costs, we measure the wall-clock time on a single A40 GPU. For the space cost, although ideally one would measure it by memory usage, this can be challenging because memory usage is sensitive to the specific implementation and parallelization. Therefore, we measure the space cost by the total parameter count instead. For Dropout Ensemble, we will also use the number of independently trained models (I) as a measure of the training time cost and space cost, as both of the two costs are proportional to I in this case. More details about the measurements are provided in Appendix D.

4.2 Dropout Ensemble

In this section, we empirically evaluate the efficiency and efficacy of the proposed Dropout Ensemble and LoRA Ensemble.

Improvement over the training time cost. To illustrate the improvement over training time cost comparing Dropout Ensemble to the naive ensemble, we first report the results on the TRAK method across four experiment settings. As can be seen in Figure 3, across all four experiment settings, increasing the number of dropout-masked passes (D) results in significant LDS improvement for a fixed number of independently trained models (I). Recall that Dropout Ensemble does not incur any additional training time cost for a fixed I. This implies that Dropout Ensemble can significantly reduce the training time cost for achieving the same level of attribution efficacy as measured by LDS. For example, Dropout Ensemble

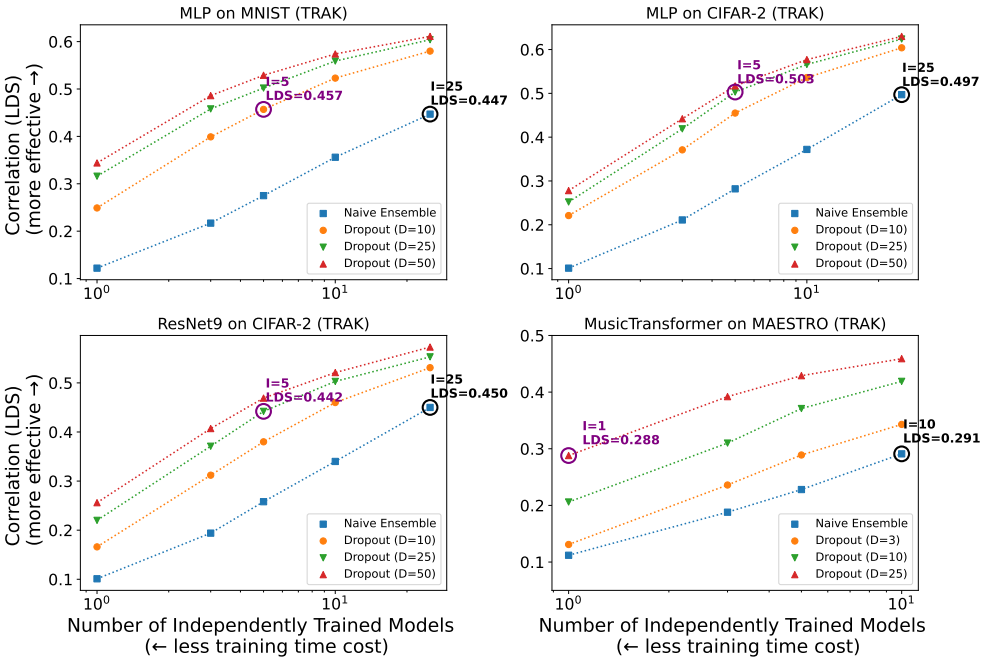


Figure 3: The LDS of naive independent ensemble and Dropout Ensemble with different numbers of dropout-masked passes (D) and independently trained models (I). We apply the ensemble methods to the TDA method, TRAK. There are four experiment settings: MLP classifiers trained on MNIST and CIFAR-2 (top row); ResNet9 trained on CIFAR-2 (bottom-left); and Music Transformer trained on MAESTRO (bottom-right). The x-axis indicates the training time and storage cost measured by the number of independently trained models (I). The y-axis indicates the attribution efficacy measured by LDS.

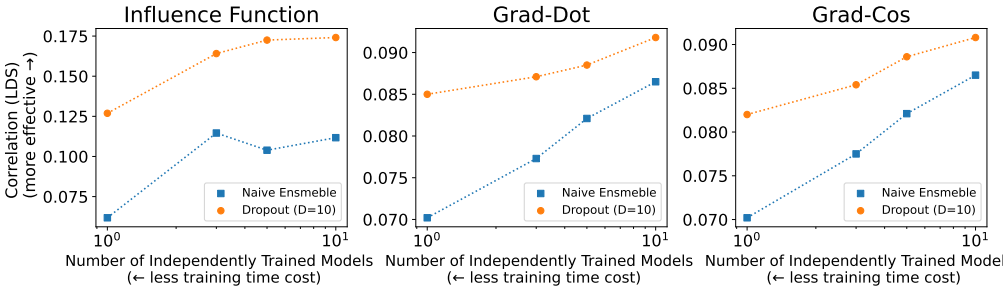


Figure 4: The LDS of naive independent ensemble and Dropout Ensemble on more TDA methods, IF, Grad-Dot, and Grad-Cos. The experiments are performed on MLP classifiers trained on MNIST. The plot setup is similar as Figure 3.

with $I = 5$ (circled by purple circle) can reach higher LDS than naive independent ensemble with $I = 25$ (circled by black circle) for MLP on MNIST and MLP/ResNet9 on CIFAR-2, which achieves a 80% reduction on training time cost. For Music Transformer on MAESTRO, Dropout Ensemble with $I = 1$ (circled by purple circle) can reach similar LDS than naive independent ensemble with $I = 10$ (circled by black circle), which achieves a 90% reduction.

We find that Dropout Ensemble works similarly well for other TDA methods. In Figure 4, we report the results on IF, Grad-Dot, and Grad-Cos. We only experiment on the setting of MLP classifiers trained on MNIST, as these TDA methods do not have meaningfully good efficacy on more complex settings. Similar to the TRAK experiments, we observe that Dropout Ensemble with $I = 1$ could match the LDS of naive independent ensemble with $I = 10$ for IF and Grad-Dot, and match that with $I = 5$ for Grad-Cos.

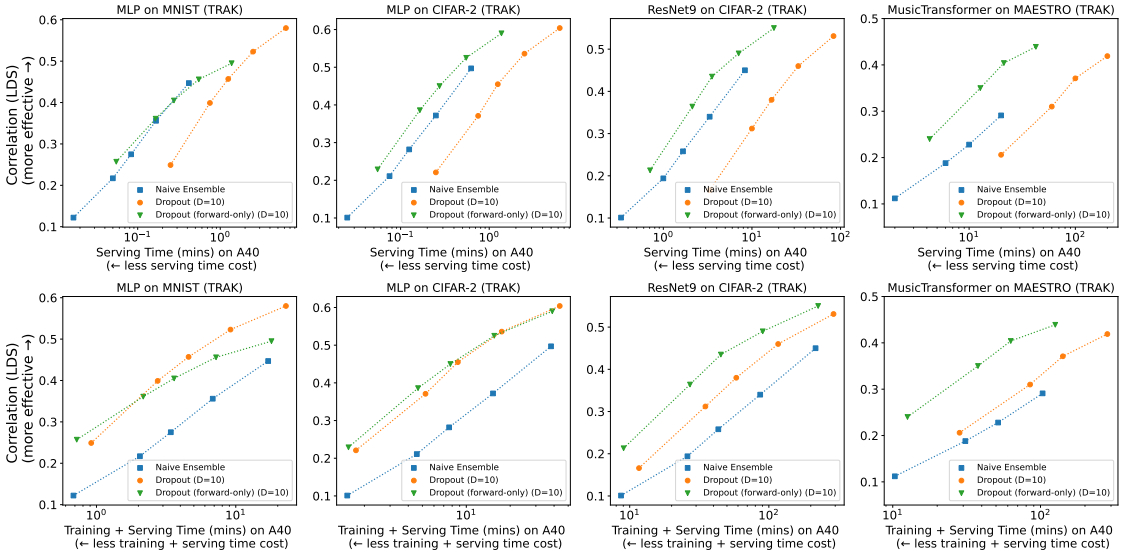


Figure 5: The LDS of Dropout Ensemble and its variant Dropout Ensemble (forward-only) when applied to TRAK on different dataset and model settings. The x-axis indicates the serving (first row) / training+serving (second row) time cost measured by the running time on a single A40 GPU. The y-axis indicates the attribution efficacy measured by LDS. The points in the plot correspond to different numbers of independent trained models (I). The number of dropout masked models (D) is fixed to 10.

Therefore, we expect that the proposed Dropout Ensemble can be generalized to various gradient-based TDA methods.

Improvement over the space cost. The improvement of Dropout Ensemble over the space cost is self-evident, which can also be measured by I, the number of independently trained models. According to Figure 3, for example, applying Dropout Ensemble on TRAK (in comparison to naive independent ensemble) can lead to an 80% or 90% reduction in space cost for different model and dataset settings.

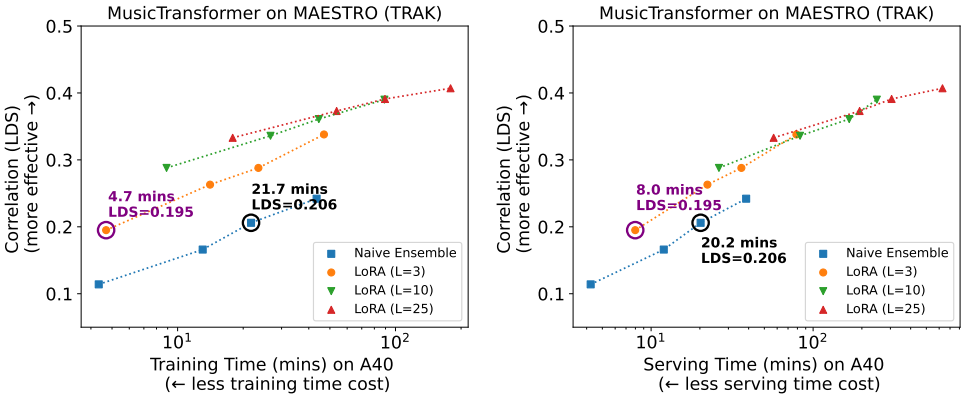
Overhead over the serving time cost and Dropout Ensemble (forward-only) improvement. There is serving time cost overhead caused by Dropout Ensemble though the overhead is relatively small compared to the significant improvement over training time cost. In this paragraph, we will also show the improvement of Dropout Ensemble (forward-only) a variant of Dropout Ensemble specifically tailored for TRAK, over serving time cost.

We record the serving time of Dropout Ensemble and Dropout Ensemble (forward-only) and plot them in the first row of Figure 5. The results show that Dropout Ensemble(forward-only) could still outperform the naive ensemble even if only serving cost is considered. Dropout Ensemble has a larger serving time cost overhead. It is worth noting that the training cost and the serving cost are both for the TDA process, which are defined respectively in Section 3.2. The computational cost of TDA should consider both costs together. We also record the training + serving time of each ensemble method and plot them in the second line of Figure 5. The result shows that Dropout Ensemble (and the forward-only variant) can achieve a better efficacy-efficiency trade-off than the naive ensemble over the train + serving time cost.

We report additional ablation studies, memory usages, performance of downstream tasks, and qualitative results of Dropout Ensemble in Appendix E. We also have a brief discussion about the surprisingly good performance of Dropout Ensemble(forward-only) in Appendix G.

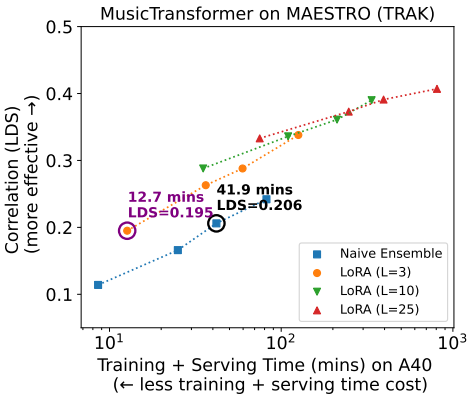
4.3 LoRA Ensemble

To demonstrate the efficiency of LoRA Ensemble, we only consider TRAK Park et al. (2023) as the main TDA method in this section since it is the most effective (in terms of LDS) among the four gradient-based TDA

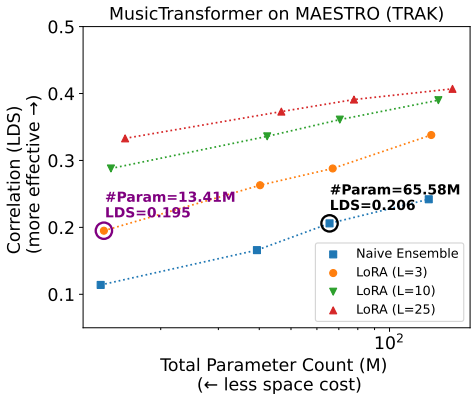


(a) Training time cost improvement.

(b) Serving time cost improvement.



(c) Total time cost improvement.



(d) Space cost improvement.

Figure 6: The LDS of naive ensemble and LoRA Ensemble with respect to different cost measurements. Here, we apply the ensemble methods to TRAK on Music Transformer trained on the MAESTRO dataset. The x-axis of Figure (6a, 6b, 6c) indicates training/serving time costs (running time on a single A40), while that of Figure 6d specifies the space cost (total parameter count). The y-axis of all figures is the attribution efficacy measured by LDS. LoRA Ensemble has significantly fewer costs in all aspects than naive ensemble for achieving similar LDS.

methods discussed in Section 4.2. We experiment LoRA Ensemble under the setting of Music Transformer on MAESTRO because this setting aligns better with large-scale real-world generative settings where LoRA adapters are more prevalent.

As shown in Figure 6, LoRA Ensemble outperforms naive independent ensemble since it can reach similar LDS as the latter with significantly less computational costs in terms of all types of measurements. For example, LoRA Ensemble with ($I = 1, L = 3$) (circled by purple circle) achieves similar LDS as naive ensemble with $I = 5$ (circled by black circle), while having reduction in training time (Fig 6a), serving time (Fig 6b), and total parameter count (Fig 6d) respectively for 78.4%, 60.5%, and 79.6%.

4.4 Large experiment results

We include additional experiments to further demonstrate the effectiveness of Dropout Ensemble and LoRA Ensemble with the implementation provided by Deng et al. (2024). Firstly, there are three new settings: (1) GPT-2 (Radford et al., 2019) on WikiText2 (Merity et al., 2016) (2) NanoGPT (Karpathy, 2022) on Shakespeare (Karpathy, 2015) for language modeling and (3) VisionTransformer (ViT) (Dosovitskiy et al., 2021) on CIFAR-10 (Krizhevsky et al., 2009) for image classification. We also consider two enlarged experiments included in Section 4.2 and 4.3: (4) MLP with 60K training samples from MNIST and (5) MusicTransformer with 15K training samples from MAESTRO. Detailed experiment setups are included in

Appendix C.1. As can be seen in Figure 7, both methods show clear improvements across new architectures and datasets.

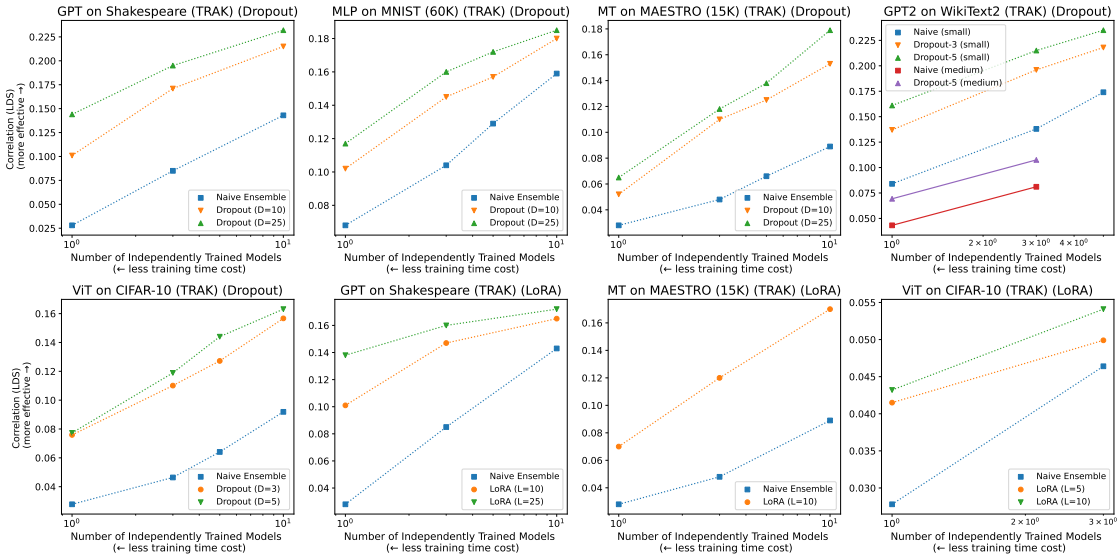


Figure 7: The LDS of Dropout Ensemble and LoRA Ensemble applied to large experiment settings. The x-axis indicates the number of independently trained models (I), the y-axis indicates the attribution efficacy measured by LDS. Starting from the top-left corner, the first five figures are conducted with Dropout Ensemble while the final three are conducted with LoRA Ensemble.

4.5 Comparison between Dropout Ensemble and LoRA Ensemble

We propose two efficient ensemble methods and we compare them in this subsection as a guide for method selection.

Efficacy. Dropout Ensemble yield slightly better improvement compared with LoRA Ensemble and both ensembles can significantly outperform naive ensembles by achieving a better efficacy-efficiency tradeoff.

Efficiency. Dropout Ensemble and LoRA Ensemble have different trade-offs regarding training and serving costs. Dropout Ensemble reduces training costs more significantly while LoRA Ensemble could reduce both training and serving time costs when they reach similar LDS compared to naive ensemble. Their suitability may vary depending on the specific applications. Dropout Ensemble is more suitable for TDA tasks where training costs account for larger ratios to serving costs.

Applicability. LoRA Ensemble is designed for large transformer-based models, so we only perform experiments on the transformer architecture, e.g., NanoGPT, ViT, and Music Transformer. Dropout Ensemble can be applied to nearly all model architectures.

To conclude, Dropout Ensemble can be seen as a go-to method for efficient ensemble of TDA and LoRA Ensemble performs as a good choice for TDA tasks on Transformer-based models.

5 Theoretical Analysis of Naive and Two-step Ensemble

In this section, we present a theoretical analysis to compare TDA method with the naive ensemble (regular independent ensemble) and our two-step ensemble (a general case containing Dropout Ensemble and LoRA Ensemble). With notation defined in Section 3.1, for any test sample $x \in \mathcal{T}$ and $\mathcal{S}$ as the training set, we assume a TDA method τ finds an "optimal" attribution score $\tau(\Theta^*)$ based on the optimal parameter Θ^* that some ensemble estimators try to approximate (omit f_Θ here for notational convenience). Then, we define the two types of ensemble estimators for $\tau(\Theta^*)$ and through Lemma 5.1 below to show that Dropout Ensemble and LoRA Ensemble outperform the regular ensemble in terms of the approximation error.

We start with the definition of the two types of ensemble estimators. For a TDA method τ, the *regular ensemble estimator* τ_{ens} is defined as the average of $\tau(\Theta^{(i)})$, where $\Theta^{(i)}$ for $i = 1, \ldots, I$ are I identically distributed (i.d.) individual estimators. The *two-step ensemble estimator* $\tau_{\text{2-step}}$ is defined as the average of $\tau(\Theta^{(k,d)})$, where each of $\Theta^{(k)}$ for $k = 1, \cdots, K$ is an individual estimator (just as $\Theta^{(i)}$ in *regular ensemble estimator*) and by using $\Theta^{(k)}$ as a base estimator, D variants of it, $\Theta^{(k,d)}$, are generated for the second-step ensemble. This is a general definition and includes Dropout Ensemble and LoRA Ensemble as special cases. Our goal is to compare the squared error of τ_{ens} and $\tau_{\text{2-step}}$ with respect to $\tau(\Theta^*)$, which we formalize as the following lemma (proof in Appendix F.1).

Lemma 5.1. *Define τ_{ens} and $\tau_{2\text{-}step}$ as the regular and two-step ensemble estimators for $\tau(\Theta^*)$:*

$$\tau_{ens} = \frac{1}{I}\sum_{i=1}^{I}\tau(\Theta^{(i)}) \quad \textit{and} \quad \tau_{2\text{-}step} = \frac{1}{KD}\sum_{k=1}^{K}\sum_{d=1}^{D}\tau(\Theta^{(k,d)}).$$

Assume each individual estimate $\tau(\Theta^{(i)})$ and $\tau(\Theta^{(k,d)})$ have the same distribution. Then, the difference in squared error $\Delta = \|\tau_{ens} - \tau(\Theta^*)\|^2 - \|\tau_{2\text{-}step} - \tau(\Theta^*)\|^2$ is given by:

$$\Delta = \frac{1}{I}\left(\Sigma_{1,1} + (I-1)\Sigma_{i,j}\right) - \frac{1}{KD}\Big(\Sigma_{(k,m),(k,m)} + (D-1)\Sigma_{(k,m),(k,n)} + D(K-1)\Sigma_{(k,m),(l,n)}\Big),$$

- *$\Sigma_{i,j} = Cov(\tau(\Theta^{(i)}), \tau(\Theta^{(j)}))$ is the covariance between individual estimators in the regular ensemble. Given the i.d. assumption, $\Sigma_{i,j}$ are the same for all $i \neq j$, and $\Sigma_{i,j} = \Sigma_{1,1}$ for $i = j$.*
- *$\Sigma_{(k,m),(l,n)} = Cov(\tau(\Theta^{(k,m)}), \tau(\Theta^{(l,n)}))$ is the covariance between variants of base individual estimators in the two-step ensemble. We further define $\Sigma_{(k,m),(k,n)}$ as the within-group covariance and $\Sigma_{(k,m),(l,n)}$ with $k \neq l$ as the between-group covariance.*

Now, we analyze Δ and show that Δ will stay positive under reasonable assumptions, which implies that $\tau_{\text{2-step}}$ outperforms τ_{ens}. We especially derive Δ for two interesting cases: $K = I$ and $KD = I$ (see assumptions and derivation in Appendix F.2).

Case 1: $K = I$ means the number of individual estimators K in the two-step ensemble is equal to the number of individual estimators I in the regular ensemble, but the two-step ensemble has D variants for each base estimator, resulting in a total of $K \cdot D = I \cdot D$ estimates. In this case, we derive

$$\Delta = \frac{D-1}{ID}\left(\Sigma_{1,1} - \Sigma_{(k,m),(k,n)}\right). \tag{3}$$

As long as the within-group covariance $\Sigma_{(k,m),(k,n)}$ is smaller than variance of each individual estimator $\Sigma_{1,1}$, $\tau_{\text{2-step}}$ will outperform the regular ensemble τ_{ens}. This is likely to be the case for any pair of different variants of the same base estimator through dropout or LoRA fine-tuning. For a fixed small I, when D is small, $\frac{D-1}{D}$ has bigger impact on Δ and increase D can quickly increase Δ means $\tau_{\text{2-step}}$ outperforms τ_{ens} more. In contrast, when D is large, $\frac{D-1}{D}$ is close to 1 and increase D will not change Δ much, meaning the performance gain of $\tau_{\text{2-step}}$ over τ_{ens} will saturate, which matches our empirical results in Figure 3 and Figure 6. On the other hand, for a large I, changing D will not change Δ much as Δ is already small, and the performance gain of $\tau_{\text{2-step}}$ over τ_{ens} will be less significant, which also aligns with our empirical observation in Figure 3 and Figure 6. In summary, when $K = I$, $\tau_{\text{2-step}}$ clearly outperforms τ_{ens} when the within-group covariance is small benefiting from averaging over more total estimators (ID versus I), and the performance gain saturates as the within-group covariance increases.

Case 2: $KD = I$ means the total number of estimators in the two-step ensemble $K \cdot D$ is equal to the number of estimators I in the regular ensemble. Thus, both ensembles have the same number of estimates, but the two-step ensemble introduces a base-variant relationship. In this case, we derive:

$$\Delta = \frac{D-1}{I}\left(\Sigma_{i,j} - \Sigma_{(k,m),(k,n)}\right). \tag{4}$$

This case is about comparing the within-group covariance $\Sigma_{(k,m),(k,n)}$ and the individual estimator covariance $\Sigma_{i,j}$. In general, $\Sigma_{(k,m),(k,n)}$ is expected to be larger than $\Sigma_{i,j}$, because $\Sigma_{(k,m),(k,n)}$ is the covariance between variants of the same base estimator, while $\Sigma_{i,j}$ is the covariance between different estimators. This can lead to negative Δ, indicating that the two-step ensemble is outperformed by the regular ensemble in this case. However, since $KD = I$, $\frac{D-1}{I}$ is close to $\frac{1}{K}$, meaning $\tau_{\text{2-step}}$ is not significantly outperformed by τ_{ens} when a large K is set. In summary, when $KD = I$, $\tau_{\text{2-step}}$ can still outperform τ_{ens} if the within-group covariance is small. However, if this covariance is large, then $\tau_{\text{2-step}}$'s advantage diminishes, and τ_{ens} may perform similarly or better due to the independent nature of its estimators. Even for the latter, two-step ensemble still enjoy drastic efficiency gain as generating more variants for each base estimator is much cheaper, e.g., via dropout or LoRA fine-tuning, than generating more individual estimators from re-training.

Empirical studies. Previous analysis shows that the general two-step ensemble can outperform the naive ensemble in efficacy for Case 1 under certain assumptions and can achieve close efficacy to the naive ensemble for Case 2 with much cheaper cost. In this paragraph, we examine the assumptions in our proposed efficient ensemble methods as well as provide empirical computation of the covariances.

Case 1 means that for both the naive ensemble and the two-step ensemble, we use the same number of independently trained models. The gap between the naive ensemble and the two-step ensemble is defined by Equation 3. Two-step ensemble is better as long as $\Sigma_{1,1} > \Sigma_{(k,m)(k,n)}$. This is likely to be the case because the variance of each estimator typically remains at the same level. The empirical result in Table 1 also shows the assumption holds. In Figure 3, Dropout Ensemble outperforms naive ensembles with the number of independently trained models fixed.

Case 2 means that the total number of estimators for the naive ensemble and the two-step ensemble are the same (but two-step ensemble methods will be much more efficient since D variants of the second-step ensemble can be acquired through Dropout or LoRA). Under this case, most settings of Dropout Ensemble and LoRA Ensemble do not outperform the naive ensemble in our previous efficacy experiments. The empirical result in Table 1 also shows that $\Sigma_{(k,m)(k,n)} > \Sigma_{i,j}$. We want to point out that case 2 is a very challenging case for two-stage ensemble methods to outperform naive ensemble in efficacy. We encourage further studies of efficient ensemble methods that could outperform naive ensembles in case 2.

	$\Sigma_{1,1}$	$\Sigma_{(k,m),(k,n)}$	$\Sigma_{i,j}$
Test sample 1	3.26e-07	1.31e-07	6.72e-08
Test sample 2	2.85e-07	1.03e-07	5.26e-08
Test sample 3	7.43e-07	2.68e-07	1.34e-07

Table 1: Covariances of different estimators of MLP on MNIST and Dropout Ensemble. $\Sigma_{1,1}$ represent the variance of each individual estimator, $\Sigma_{(k,m),(k,n)}$ represent the within-group covariance, and $\Sigma_{i,j}$ represent the covariance between different estimators. We randomly choose three test samples for the emprical analysis of covariance. Among the selected test samples, $\Sigma_{1,1} > \Sigma_{(k,m)(k,n)} > \Sigma_{i,j}$.

6 Conclusion

We present Dropout Ensemble and LoRA Ensemble as efficient alternatives to the naive independent ensemble approach for improving gradient-based TDA methods. The proposed strategies significantly reduce training time (up to 80%), serving time (up to 60%), and space cost (up to 80%), while maintaining similar attribution efficacy in comparison to the naive ensemble. Empirical results from our extensive experiments show that the proposed efficient ensembles can remarkably advance the Pareto frontier of TDA methods with better computational efficiency and TDA efficacy. Notably, we demonstrate the proposed methods work well on several generative modeling settings. Our theoretical analysis shows our ensemble method has a smaller squared error for estimating the optimal scores compared to naive ensembles. In the future, we will utilize our methods in more real-world generative settings that were blocked by high computational costs or low effectiveness of TDA methods.

References

Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger B Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022.

Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. Relatif: Identifying explanatory training samples via relative influence. In *International Conference on Artificial Intelligence and Statistics*, pp. 1899–1909. PMLR, 2020.

Samyadeep Basu, Philip Pope, and Soheil Feizi. Influence functions in deep learning are fragile. *arXiv preprint arXiv:2006.14651*, 2020.

Guillaume Charpiat, Nicolas Girard, Loris Felardos, and Yuliya Tarabalka. Input similarity from the neural network perspective. *Advances in Neural Information Processing Systems*, 32, 2019.

Junwei Deng and Jiaqi Ma. Computational copyright: Towards a royalty model for ai music generation platforms. *arXiv preprint arXiv:2312.06646*, 2023.

Junwei Deng, Ting-Wei Li, Shiyuan Zhang, Shixuan Liu, Yijun Pan, Hao Huang, Xinhe Wang, Pingbang Hu, Xingjian Zhang, et al. dattri: A library for efficient data attribution. *Advances in Neural Information Processing Systems*, 37:136763–136781, 2024.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=YicbFdNTTy`.

Logan Engstrom, Axel Feldmann, and Aleksander Madry. Dsdm: Model-aware dataset selection with datamodels. *arXiv preprint arXiv:2401.12926*, 2024.

Vitaly Feldman and Chiyuan Zhang. What neural networks memorize and why: Discovering the long tail via influence estimation. *Advances in Neural Information Processing Systems*, 33:2881–2891, 2020.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pp. 2242–2251. PMLR, 2019.

Han Guo, Nazneen Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. FastIF: Scalable influence functions for efficient model interpretation and debugging. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.808.

Zayd Hammoudeh and Daniel Lowd. Training data influence analysis and estimation: A survey. *Machine Learning*, pp. 1–53, 2024.

Curtis Hawthorne, Andriy Stasyuk, Adam Roberts, Ian Simon, Cheng-Zhi Anna Huang, Sander Dieleman, Erich Elsen, Jesse Engel, and Douglas Eck. Enabling factorized piano music modeling and generation with the maestro dataset. *arXiv preprint arXiv:1810.12247*, 2018.

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.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Curtis Hawthorne, Andrew M Dai, Matthew D Hoffman, and Douglas Eck. Music transformer: Generating music with long-term structure. *arXiv preprint arXiv:1809.04281*, 2018.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

Ruoxi Jia, David Dao, Boxin Wang, Frances Ann Hubis, Nick Hynes, Nezihe Merve Gürel, Bo Li, Ce Zhang, Dawn Song, and Costas J Spanos. Towards efficient data valuation based on the shapley value. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1167–1176. PMLR, 2019.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Andrej Karpathy. char-rnn. `https://github.com/karpathy/char-rnn`, 2015.

Andrej Karpathy. nano-gpt. `https://github.com/karpathy/nanoGPT`, 2022.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

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

Yongchan Kwon and James Zou. Beta shapley: a unified and noise-reduced data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 8780–8802. PMLR, 2022.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. *arXiv preprint arXiv:2310.00902*, 2023.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Dan Ley, Suraj Srinivas, Shichang Zhang, Gili Rusak, and Himabindu Lakkaraju. Generalized group data attribution. *arXiv preprint arXiv:2410.09940*, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models, 2016.

Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. Trak: Attributing model behavior at scale. *arXiv preprint arXiv:2303.14186*, 2023.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8179–8186, 2022.

Anders Søgaard et al. Revisiting methods for finding influential examples. *arXiv preprint arXiv:2111.04683*, 2021.

C. Spearman. The proof and measurement of association between two things. *The American Journal of Psychology*, 15(1):72–101, 1904.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1): 1929–1958, 2014.

John Tukey. Bias and confidence in not quite large samples. *Ann. Math. Statist.*, 29:614, 1958.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Jiachen T Wang and Ruoxi Jia. Data banzhaf: A robust data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 6388–6421. PMLR, 2023.

Xi Wang, Laurence Aitchison, and Maja Rudolph. Lora ensembles for large language model fine-tuning. *arXiv preprint arXiv:2310.00035*, 2023.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

Chih-Kuan Yeh, Joon Kim, Ian En-Hsu Yen, and Pradeep K Ravikumar. Representer point selection for explaining deep neural networks. *Advances in neural information processing systems*, 31, 2018.

Appendix

A Related work (cont'd) **17**

B TDA methods and metrics **17**

B.1 TDA methods . . . 17

B.2 Linear datamodeling score (LDS) . . . 18

C Detailed experiment setup **19**

C.1 Datasets and models . . . 19

C.2 Efficient ensemble setup . . . 20

D Wall-clock time measurements **20**

D.1 Quantitative analysis of computation overhead . . . 21

E Additional experiments **22**

E.1 Ablation experiment to random projection . . . 22

E.2 Intermediate checkpoints . . . 22

E.3 Stochastic Weight Averaging (SWA) . . . 22

E.4 Memory costs of Dropout Ensemble and LoRA Ensemble . . . 23

E.5 Space costs of Dropout Ensemble and LoRA Ensemble . . . 23

E.6 Qualitative results . . . 24

E.7 Noisy label detection results . . . 24

E.8 Data selection results . . . 24

E.9 Hyperparameter sensitivity . . . 24

F Proofs **25**

F.1 Proof of Lemma 5.1 . . . 25

F.2 Derivation of the error difference for $K = I$ and $KD = I$ . . . 27

G Discussion of Dropout Ensemble (forward-only) **28**

A Related work (cont'd)

Retraining-based TDA methods. Retraining-based methods compute TDA scores by systematically retraining the model with and without specific training samples to quantify their influence on the model's output. These methods are computationally expensive due to the requirement of the large amount of model retrainings. For example, Leave-One-Out (LOO) influence Tukey (1958) measures the prediction difference between the model trained on the full training dataset and models trained on subsets with only one specific sample dropped. Data Shapley (Ghorbani & Zou, 2019; Jia et al., 2019), Beta-Shapley (Kwon & Zou, 2022), and Data Banzhaf (Wang & Jia, 2023) extends the LOO idea to consider data interactions for more equitable TDA scores, but they require retraining models on all possible subsets of the training dataset. Similarly, DataModels (Ilyas et al., 2022) tries to learn the model predictions when the model is trained on each subset of the training dataset, which requires a nontrivial amount of model retraining. While, in practice, sampling or approximation will be used to reduce the number of model retrainings, these methods typically still require thousands of or more retrainings to achieve satisfactory attribution efficacy. In summary, the high demand for retraining makes these TDA methods computationally inefficient and limits their practical applicability to even moderately large models.

Gradient-based TDA methods. Gradient-based methods are another group of TDA methods that usually provide closed-form TDA scores using gradients. Since the seminal work of influence function by Koh & Liang (2017), gradient-based methods have become increasingly popular due to their scalability. The influence function (Koh & Liang, 2017) and subsequent studies (Guo et al., 2021; Barshan et al., 2020; Schioppa et al., 2022; Kwon et al., 2023) obtain TDA scores by approximating the effect of upweighting a training sample on the loss function. Moreover, Representer Point Selection (Yeh et al., 2018) decomposes the pre-activation of a neural network as a linear combination of training samples. TracIn (Pruthi et al., 2020) traces the loss changes on the test points during the training process. TRAK (Park et al., 2023) uses the neural tangent kernel with random projection to assess influence. These gradient-based methods significantly reduced the computational cost compared to retraining-based methods. The downside is that they typically rely on the convexity assumption and Taylor approximation to calculate TDA scores. These requirements lead to performance degradation on non-convex neural networks and sensitivity to the randomness inherent in model initialization and training. To improve the efficiency of these TDA methods, grouping training data has been explored (Ley et al., 2024).

Dropout and LoRA in ensembling. There have been some studies using dropout (Gal & Ghahramani, 2016) and LoRA (Wang et al., 2023) in ensembling, though not connected with TDA. To the best of our knowledge, this is the first study to propose an alternative to fully independent training to achieve a better balance between effectiveness and efficiency in TDA ensembling. Additionally, it uncovers intriguing links between dropout, LoRA fine-tuning, and ensembling within the TDA framework. Moving beyond conventional ensembling is both an interesting and practically valuable discovery.

B TDA methods and metrics

B.1 TDA methods

In this section, we provide details on the TDA methods we perform efficient ensembling on. Additionally, we also introduce the ensembling aggregation methods, i.e., the way to aggregate $\tau_{\text{ens}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^{I})$, for each method. The notation will follow Section 3.1.

Tracing with the Randomly-projected After Kernel (TRAK). This is a state-of-the-art gradient-based TDA method provided by Park et al. (2023). It natively introduces ensembling in its definition.

$$\tau_{\text{TRAK}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^{I}) = \left(\frac{1}{I} \sum_{i=1}^{I} \mathbf{Q}_{f_{\Theta^{(i)}}} \right) \left(\frac{1}{I} \sum_{i=1}^{I} \phi_{f_{\Theta^{(i)}}} \left(\Phi_{f_{\Theta^{(i)}}}^{\top} \Phi_{f_{\Theta^{(i)}}} \right)^{-1} \Phi_{f_{\Theta^{(i)}}}^{\top} \right),$$

Detailed notation description is described in Section 3.1. We use 2048 as the random projection dimension for TRAK. For each independently trained model, we train the model on half sampled training set following Park et al. (2023).

Influence functions based on the conjugate gradients (IF). First proposed by Koh & Liang (2017), the definition of IF is

$$\tau_{\text{IF}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^{I}) = \left[\frac{1}{I} \sum_{i=1}^{I} g_{f_{\Theta^{(i)}}}(x_j)^\top H_{f_{\Theta^{(i)}}}^{-1} g_{f_{\Theta^{(i)}}}(x) : x_j \in \mathcal{S} \right],$$

where $g_{f_{\Theta^{(i)}}}(x)$ is the vector gradient of model output $f(x; \Theta^{(i)})$ with respect to the parameters $\Theta^{(i)}$, $H_{f_{\Theta^{(i)}}}^{-1}$ is the inverse hessian matrix with respect to the training set, and $g_{f_{\Theta^{(i)}}}(x_j)^\top$ is the vector gradient to the training sample. The product of the first two terms are an inverse-hessian-vector-product problem, we implement conjugate gradients approach to solve it.

Grad-Dot. Grad-Dot is proposed by Charpiat et al. (2019). We simply calculate Grad-Dot multiple times on trained parameters $\Theta^{(i)}, i \in \{1, \ldots, K\}$ and take the average value.

$$\tau_{\text{Grad-Dot}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^{I}) = \left[\frac{1}{I} \sum_{i=1}^{I} g_{f_{\Theta^{(i)}}}(x)^\top g_{f_{\Theta^{(i)}}}(x_j) : x_j \in \mathcal{S} \right],$$

where $g_{f_{\Theta^{(i)}}}(x)^\top$ is the vector gradient of model output $f(x; \Theta^{(i)})$ with respect to the parameters $\Theta^{(i)}$, and $g_{f_{\Theta^{(i)}}}(x_j)^\top$ is the gradient to the training sample.

Grad-Cos. Similar to Grad-Dot,

$$\tau_{\text{Grad-Cos}}(x, \mathcal{S}; \{f_{\Theta^{(i)}}\}_{i=1}^{I}) = \left[\frac{1}{I} \sum_{i=1}^{I} \frac{g_{f_{\Theta^{(i)}}}(x)^\top}{\|g_{f_{\Theta^{(i)}}}(x)\|} \frac{g_{f_{\Theta^{(i)}}}(x_j)}{\|g_{f_{\Theta^{(i)}}}(x_j)\|} : x_j \in \mathcal{S} \right],$$

where $g_{f_{\Theta^{(i)}}}(x)^\top$ is the vector gradient of model output $f(x; \Theta^{(i)})$ with respect to the parameters $\Theta^{(i)}$, and $g_{f_{\Theta^{(i)}}}(x_j)^\top$ is the gradient to the training sample.

B.2 Linear datamodeling score (LDS)

Park et al. (2023) proposed the *linear datamodeling score* (LDS), aiming at probing the TDA method's ability to make counterfactual predictions based on the attribution score derived from the learned model output function f_Θ and the corresponding dataset to train f_Θ. Because most TDA methods are assumed to be *additive*[2], the TDA scores can be used to predict the model output function learned from a subset of training data in a summation form. Formally, the *attribution-based output predictions* of the model output function $f_{\Theta_{\mathcal{S}'}}$ is defined as follows:

$$g_\tau(x, \mathcal{S}'; \mathcal{S}) \triangleq \sum_{i: x_i \in \mathcal{S}'} \tau(x, \mathcal{S}; f_\Theta)_i, \tag{5}$$

where $\mathcal{S}$ is the training set, $\mathcal{S}' \subseteq \mathcal{S}$ is a subset of $\mathcal{S}$ and $f_{\Theta_{\mathcal{S}'}}$ is the model output function with $\Theta_{\mathcal{S}'}$ learned from $\mathcal{S}'$. Intuitively, $g_\tau(x, \mathcal{S}'; \mathcal{S})$ computes the overall attribution of the subset $\mathcal{S}'$ on example x, which should be a powerful indicator of the model prediction on x (i.e., $f_{\Theta_{\mathcal{S}'}}(x)$) if the TDA method works well. The *linear datamodeling score* (LDS) is defined to measure the predictive power of $g_\tau(x, \mathcal{S}'; \mathcal{S})$ and can be formalized as follows:

Definition B.1 (Linear datamodeling score)**.** Given a training set $\mathcal{S}$, a model output function f_Θ, and a corresponding TDA method τ. Let $\{\mathcal{S}_1, \ldots, \mathcal{S}_m : \mathcal{S}_j \subseteq \mathcal{S}\}$ be m randomly sampled subsets of $\mathcal{S}$, each of size

[2]If a TDA method is additive, then it defines an attribution score that the overall influence of a group is the sum of the individual influence in the group.

$\alpha \times n$ for some fixed $\alpha \in (0, 1)$. The *linear datamodeling score* (LDS) of τ for a specific example x is defined as

$$LDS(\tau, x) \triangleq \boldsymbol{\rho}(\{f_{\Theta_{\mathcal{S}_j}}(x) : j \in [m]\}, \{g_\tau(x, \mathcal{S}_j; \mathcal{S}) : j \in [m]\}),$$

where $\boldsymbol{\rho}$ is the Spearman rank correlation (Spearman, 1904), $f_{\Theta_{\mathcal{S}_j}}$ is the model output function with $\Theta_{\mathcal{S}_j}$ learned from $\mathcal{S}_j$ and $g_\tau(x, \mathcal{S}_j; \mathcal{S})$ is defined in Eq (5).

To compute LDS for our experiment settings, we use 50 models that are independently trained on random subsets with size half of the full dataset (i.e., we set $m = 50$ and $\alpha = 0.5$ in Definition B.1). We report the mean LDS on all test samples $x \in \mathcal{T}$. The LDS ranges from -1.0 to 1.0, where a value of 1.0 represents the ideal result. While this ideal is nearly impossible to achieve in practice, it can be approached with a large number of ensembles.

C Detailed experiment setup

C.1 Datasets and models

MLP on MNIST-10. For the MNIST-10 (LeCun et al., 1998) experiment, we sampled 5000 training samples and 500 testing samples. We used a 3-layer MLP with hidden layer sizes equal to 128 and 64 and placed dropout layers after the first two linear layers with a rate of 0.1, which resulted in a total of about 0.11M parameters. We employed an SGD optimizer with learning rate 0.01, momentum 0.9, and batch size 64 to train this MLP classifier for 100 epochs.

MLP/ResNet-9 on CIFAR-2. For CIFAR-2 experiment, we construct the CIFAR-2 dataset by sampling from the CIFAR-10 dataset that only include the "cat" and "dog" classes (same as the setting in Park et al. (2023)). To incorporate a diverse set of model architectures, we train an MLP and a CNN-based model on this dataset. We also only consider a subset of the CIFAR-2 dataset with a training size equal to 5000 and a testing size equal to 500. Firstly, we use a 3-layer MLP with hidden layer sizes equal to 120 and 84 and place dropout layers after the first two linear layers with a rate of 0.1, which results in a total of about 0.38M parameters. We employ an SGD optimizer with a learning rate of 0.01, momentum of 0.9, and batch size 64 to train this MLP classifier for 50 epochs. Additionally, we consider a standard ResNet-9 model (He et al., 2016) with dropout layers being placed after all convolution layers, which has roughly 4.83M trainable parameters. We train this model for 50 epochs.

Music Transformer on MAESTRO. For the MAESTRO experiment, we use the MIDI and Audio Edited for Synchronous TRacks and Organization (MAESTRO) dataset (v2.0.0) Hawthorne et al. (2018) and construct a Music Transformer following the original setting in (Huang et al., 2018). Specifically, the number of layers equals to 6, the number of independent heads equal to 8, the input feature size is 512 and the dimension of the feedforward network is 1024. For data processing, we follow the basic experiment setup used by (Deng & Ma, 2023). To be more specific, we define a vocabulary set of size equal to 388, which includes "NOTE ON" and "NOTE OFF" events for 128 different pitches, 100 "TIME SHIFT" events, and 32 "VELOCITY" events. The raw data is pre-processed as sequences of about 90K events. Due to computational constraints, we train a Music Transformer model on a subset of the official training set in the MAESTRO dataset with a size of 5000. For training, the batch size has been set to 64, and the model is trained by a classic seq2seq loss function. We employ an Adam optimizer with a learning rate equal to 1e-4, $\beta_1 = 0.9$ and $\beta_2 = 0.98$. We apply zero warm-up steps since the dataset size is comparably small, and we train the model for 20 epochs. For music event generation, we use 178 samples from the official testing dataset as prompts to generate music with a single event, which is used to evaluate the TDA methods.

ViT on CIFAR-10. For this experiment, we construct a CIFAR-10 Krizhevsky et al. (2009) split by uniformly sampling 500 images per class for training (5000 in total) and 50 per class for testing (500 in total). We adopt a Vision Transformer (ViT) Dosovitskiy et al. (2021) variant with 4 transformer layers, hidden size 256, 4 attention heads, feed-forward layer dimensionality 768, which results in 2.88 million parameters in total. We train this model with the AdamW optimizer (learning rate 1e-3, weight decay 2e-4), a batch size of 512, and run for 70 epochs, applying a simple data pipeline of 224×224 resize, patch size 16 and normalization.

NanoGPT on Shakespeare. For this experiment, we use the Shakespeare dataset (Karpathy, 2015) and train a NanoGPT model (Karpathy, 2022) on it. The model contains 6 transformer layers, and the number of multi-heads is set to 6. We process the dataset with block size 256, which generates 3921 samples. During the training, the batch size is set to 32. We train the model with the Adam optimizer with a learning rate equal to 1e-3, $\beta_1 = 0.9$, and $\beta_2 = 0.99$. The training follows the default setting of Karpathy (2022).

GPT-2 on WikiText2. For this experiment, we train GPT-2 (Radford et al., 2019) on the WikiText2 (Merity et al., 2016) dataset. The GPT-2 is the official version with 124M (small) and 355M (medium) parameters, and the dataset version is wikitext-2-raw-v1. We train the model with the AdamW optimizer with a learning rate of 5e-5 for 3 epochs.

C.2 Efficient ensemble setup

Dropout Ensemble. Dropout is applied to the same layers as the model training stated in Appendix C.1. Dropout rate is set to 0.1 for all experiment settings.

LoRA Ensemble. LoRA is applied to models with Transformer architecture only. We detail the settings for MusicTransformer and ViT as follows.

- **Music Transformer on MAESTRO.** We first train I independent models on the full training dataset with 10 epochs using different model initialization, and all the other remaining settings follow Appendix C.1. The LoRA adapters used in this experiment all have rank $r = 8$, alpha $\alpha = 8$, and trainable biases. No dropouts are applied for LoRA parameters. We augment LoRA adapters to only the W_q and W_v matrices in the self-attention modules within all the layers of the MusicTransformer. Then, we define fine-tuning datasets for each LoRA adapter as random training data subsets with a size equal to 2500 (i.e., half of the original training dataset size). We further fine-tune each of the aforementioned I independent models using L different LoRA adapters for an additional 10 epochs on these random training data subsets, which results in a total of $I \cdot L$ LoRA fine-tuned models.
- **NanoGPT on Shakespeare.** We first train I independent models on the full training dataset with 10 epochs using different model initialization. The LoRA adapters used in this experiment all have rank $r = 8$, alpha $\alpha = 8$, and trainable biases. No dropouts are applied for LoRA parameters. We augment LoRA adapters to only the W_q and W_v matrices in the self-attention modules within all the layers of the NanoGPT. Then, we define fine-tuning datasets for each LoRA adapter as random training data subsets with half size of the full training dataset. We further fine-tune each of the aforementioned I independent models using L different LoRA adapters for an additional 10 epochs on these random training data subsets, which results in a total of $I \cdot L$ LoRA fine-tuned models.
- **ViT on CIFAR-10.** We train I independent ViT models for 30 epochs on the full training set using the same AdamW schedule and the data pipeline described above. Next, for each of these base models, we inject LoRA adapters of rank $r = 8$ and LoRA scaling $\alpha = 8$ (with trainable biases only) into every query/value matrix within all transformer layers, leaving the rest of the ViT parameters frozen. For each base model, we then draw L random half-subsets of the full training data and fine-tune each LoRA-augmented model for an additional 30 epochs (with AdamW, learning rate 5e-3 and weight decay 1e-5).

D Wall-clock time measurements

In this paper, we use the wall-clock time on a single A40 GPU to measure the computational costs if the number of independently trained models (i.e., the value of I) can not precisely demonstrate the costs, e.g., the training time cost and serving time cost of LoRA Ensemble.

Here, we define several components that dominate the wall-clock time of different ensemble methods.

- $\mathrm{T}_{\text{Train}}$: The time to train a model from scratch.
- $\mathrm{T}_{\text{Train, Base}}$: The time to train a base model from scratch for LoRA tuning. Normally speaking, $\mathrm{T}_{\text{Train, Base}} < \mathrm{T}_{\text{Train}}$.
- $\mathrm{T}_{\text{Train, LoRA}}$: The time to fine-tune for one LoRA adapter.

- $\mathrm{T}_{\text{Serving}}$: The time to calculate the TDA scores for one trained model *after model training*.
- $\mathrm{T}_{\text{Serving, Forward-only}}$: The time to calculate the TDA scores for one trained models *after model training* with shared gradients. (will only be used by Dropout Ensemble(forward only). Normally speaking, $\mathrm{T}_{\text{Serving, Forward-only}} < \mathrm{T}_{\text{Serving}}$.
- $\mathrm{T}_{\text{Serving, LoRA}}$: The time to calculate the TDA scores *after model training* for one LoRA adapter. Normally speaking, $\mathrm{T}_{\text{Serving, LoRA}} < \mathrm{T}_{\text{Serving}}$.

The total computational cost is approximated and summarized in t:

- Training time cost (naive independent ensemble/Dropout Ensemble/Dropout Ensemble (forward only)): $I\times \mathrm{T}_{\text{Train}}$.
- Training time cost (LoRA Ensemble): $I\times \mathrm{T}_{\text{Train, Base}} + I \times L\times \mathrm{T}_{\text{Train, LoRA}}$.
- Serving time cost (naive independent ensemble): $I\times \mathrm{T}_{\text{Serving}}$.
- Serving time cost (Dropout Ensemble): $I \times D\times \mathrm{T}_{\text{Serving}}$.
- Serving time cost (Dropout Ensemble (forward only)): $I\times \mathrm{T}_{\text{Serving}} + I \times (D-1)\times \mathrm{T}_{\text{Serving, Forward-only}}$.
- Serving time cost (LoRA Ensemble): $I \times L\times \mathrm{T}_{\text{Serving, LoRA}}$.

D.1 Quantitative analysis of computation overhead

Quantitative analysis of Dropout Ensemble and LoRA Ensemble. As we stated in Section 3.3 and Section 3.4, computation overhead is caused by Dropout Ensemble and LoRA Ensemble compared to a naive independent ensemble. In this section, we list and record the training and serving time of each ensemble method to show a comprehensive comparison.

In Table 2, we list the computational overhead by different ensemble methods measured on MusicTransformer with TRAK as an example. The overhead of naive ensemble is proportional to the number of models used in the ensemble. In comparison, the Dropout Ensemble does not incur any additional cost in terms of training time cost (no additional training) and space cost (the dropout-masked variants do not need to be stored), while incurring additional serving time cost. The LoRA Ensemble incurs additional costs in terms of all three types of costs, but they are much smaller than naive ensemble. Furthermore, the serving time cost is significantly reduced in comparison to Dropout Ensemble since we only need to calculate the gradients with respect to the LoRA parameters.

Cost Type	No Ensemble	Naive (3)	Naive (10)	D=3	I=3, D=3	D=10	L=3	I=3, L=3	L=10
Training Time	1	3	10	1	3	1	2.00	6.00	5.50
Serving Time	1	3	10	3	9	10	1.85	5.55	6.07
Space	1	3	10	1	3	1	1.03	3.09	1.10

Table 2: The relative computation overhead for Naive Ensemble (varying number of ensembles), Dropout Ensemble and LoRA Ensemble. The "No Ensemble" column refers to the cost with only one model, where the entries are normalized to one for easier comparison. The relative costs of Naive Ensemble and Dropout Ensemble are based on simple counting, while the relative costs of LoRA Ensemble are based on experiments on MusicTransformer with TRAK.

The relationship between training and serving time cost. Here we report the computational cost of each efficient ensemble method under fixed test sets as stated in the experiment settings in Appendix C.1. The results of the ratio of training over serving cost (i.e. $\frac{\text{training time cost}}{\text{serving time cost}}$) shown in Table 3 demonstrate that training cost dominates the overall computational cost of TDA method.

Remarkably, a larger number of test samples may cause the serving time cost to exceed the training cost. Table 4 shows how many test samples are required to make the serving time cost exceed the serving time most. In typical use cases such as data selection, the test set is often smaller than the training set.

Experiment Setting	Training/Serving	Training/Serving(forward-only)
MNIST-10 + MLP	40×	200×
CIFAR-2 + MLP	60×	450×
CIFAR-2 + ResNet9	25×	208×
MAESTRO + MusicTransformer	4.2×	42×

Table 3: The training cost / serving cost ratio of each experiment settings.

Experiment Setting	Serving	Training
MNIST-10 + MLP	190K	5K
CIFAR-2 + MLP	245K	5K
CIFAR-2 + ResNet9	120K	5K
MAESTRO + MusicTransformer	17K	5K

Table 4: Number of test samples to make the serving cost exceed the training cost .

E Additional experiments

E.1 Ablation experiment to random projection

Here, we examine the root of the Dropout Ensemble's performance through an ablation experiment. There are no random factors other than dropout for IF, Grad-Dot, or Grad-Cos, while there is another random factor, i.e., random projection, in TRAK. We perform D dropout-masked passes with dropout enabled, i.e., "Dropout Ensemble" and D dropout-masked passes with dropout disabled, i.e., "Only Random Projection", in Table 5 and calculate the LDS. Random projection does contribute to the accuracy improvement, but the improvement will saturate when D is large.

	D \ I	1	3	5
Naive Independent Ensemble	N/A	0.122	0.217	0.275
Dropout Ensemble	10	**0.249**	**0.399**	**0.457**
Only Random Projection	10	0.210	0.335	0.398
Dropout Ensemble	25	**0.316**	**0.458**	**0.502**
Only Random Projection	25	0.217	0.351	0.413

Table 5: Ablating the contribution of test-time dropout. The experiment is carried out on MNIST+MLP. Note that D is the number of dropout-masked passes, and I is the number of independently trained models.

E.2 Intermediate checkpoints

Here, we show that Dropout Ensemble ensemble can also be applied to intermediate checkpoints and improve the accuracy in Table 6. Park et al. (2023) states that intermediate checkpoints with fewer epochs can be used for ensembling to reduce the training time cost. We save multiple checkpoints from different epochs of the same training process ($I = 1$).

E.3 Stochastic Weight Averaging (SWA)

Here, we show that Dropout Ensemble outperforms cheap ensemble methods such as Stochastic Weight Averaging (SWA) in Table 7. For SWA, we average the last few checkpoints of a training process.

Ensembling methods	D \ #ckpts	1	3	5	10
Naive Independent Ensemble	N/A	0.122	0.170	0.188	0.201
Dropout Ensemble	10	0.249	0.318	0.342	0.362
	25	0.316	0.359	0.373	0.389

Table 6: The TDA efficacy of Dropout Ensemble on intermediate checkpoints from different epochs of the same training process ($I = 1$). Note that D is the number of dropout-masked passes, and #ckpts is the number of intermediate checkpoints used.

	D \ I	1	3	5
Naive Independent Ensemble	N/A	0.122	0.217	0.275
Dropout Ensemble	10	**0.249**	**0.399**	**0.457**
SWA		0.121	0.231	0.291

Table 7: The TDA efficacy of Dropout Ensemble and SWA. Note that D is the number of dropout-masked passes or the number of checkpoints of SWA, and I is the number of independently trained models.

E.4 Memory costs of Dropout Ensemble and LoRA Ensemble

Here we record the peak memory usage at serving time for Dropout Ensemble (Table 8) and LoRA Ensemble (Table 9) applied on TRAK algorithm. The memory of vanilla Dropout Ensemble is the same as naive independent ensemble. The meomory of Dropout Ensemble (Forward Only) is larger than vanilla Dropout Ensemble because some of the cached terms. The memory usage of LoRA Ensemble is slightly lower than naive independent ensemble because of the reduction in parameter size.

Datasets and Models	method variants \ D	3	10	25
MNIST+MLP	Dropout Ensemble		342M	
	Dropout Ensemble (Forward Only)	636M	1956M	4787M
CIFAR2+MLP	Dropout Ensemble		344M	
	Dropout Ensemble (Forward Only)	638M	1966M	4797M
CIFAR2+ResNet9	Dropout Ensemble		477M	
	Dropout Ensemble (Forward Only)	785M	2087M	4877M
MAESTRO+MusicTransformer	Dropout Ensemble		538M	
	Dropout Ensemble (Forward Only)	634M	1529M	3421M

Table 8: The peak memory usage of Dropout Ensemble and its alternative on TRAK with different numbers of dropout-masked passes (D) and the number of independently trained models fixed to 5 ($I = 5$).

E.5 Space costs of Dropout Ensemble and LoRA Ensemble

Here we record the parameter count to present the space cost for Dropout Ensemble (Table 10) and LoRA Ensemble (Table 11).

Ensembling Methods	L \ I	1	3	5	10
Naive Independent Ensemble	N/A	431M	538M	538M	538M
LoRA Ensemble	3	404M	404M	404M	404M
LoRA Ensemble	10	404M	404M	404M	404M
LoRA Ensemble	25	404M	404M	404M	404M

Table 9: The peak memory usage of LoRA Ensemble and naive independent ensemble applied on TRAK for MusicTransformer trained on MAESTRO dataset. Note that I represents the number of independently trained models, and L is the number of LoRA adapters augmented on each model.

settings / I(w/any D)	MNIST +MLP	CIFAR-2 +MLP	CIFAR-2 +ResNet-9	MAESTRO +MusicTransformer
1	0.11M	0.38M	4.83M	13.11M
3	0.33M	1.14M	14.48M	39.35M
5	0.55M	1.89M	24.13M	65.58M
10	1.09M	3.79M	48.25M	131.16M
25	2.73M	9.48M	120.63M	327.89M

Table 10: The space cost (total parameter count) of different experiment settings under different numbers of independently trained models (I). Note that Dropout Ensemble will **not** incur any additional storage cost with more dropout-masked passes (i.e., the space cost is fixed with respect to D).

E.6 Qualitative results

We provide a qualitative results to show how Dropout Ensemble improve the TDA result without independently training more models in Figure 8. We select two random test images from MNIST-10 and the corresponding training samples that are most helpful (with highest score) and most detracting (with lowest score). The result calculated by Dropout Ensemble (D=50) is more reasonable by manual inspection. We can observe that the training images identified by Dropout Ensemble (D=50) have "shapes" similar to the corresponding test images while helpful (detracting) examples are of the same (different) class as the test images themselves.

E.7 Noisy label detection results

The improvement of efficient ensembles could also be demonstrated in downstream tasks of TDA, such as noisy label detection. We present the AUC of noisy label detection on MNIST-10 with 20% randomly flipped training samples in Table 12. The performance of TRAK with Dropout Ensemble outperforms the naive independent ensemble without training more models.

E.8 Data selection results

The improvement of efficient ensembles could also be demonstrated in downstream tasks of TDA, such as data selection. We present the test accuracy of on MNIST-10 with 10% randomly flipped training samples in Table 13. The performance of TRAK with Dropout Ensemble outperforms the naive independent ensemble without training more models.

E.9 Hyperparameter sensitivity

To illustrate the robustness of the two ensembles, we conduct hyperparameter analysis as follows.

I \ L	Naive Independent ensemble ($L = 0$)	3	10	25
1	13.11M	13.41M	14.09M	15.57M
3	39.35M	40.23M	42.29M	46.72M
5	65.58M	67.05M	70.49M	77.87M
10	131.76M	134.11M	140.99M	155.73M

Table 11: The space cost (total parameter count) of MusicTransformer under different numbers of independently trained models (I) and different numbers of LoRA adapters (L). LoRA Ensemble only add marginal space cost (at most a 18.7% increment for $D = 25$ across all I) compared to naive independent ensembling.

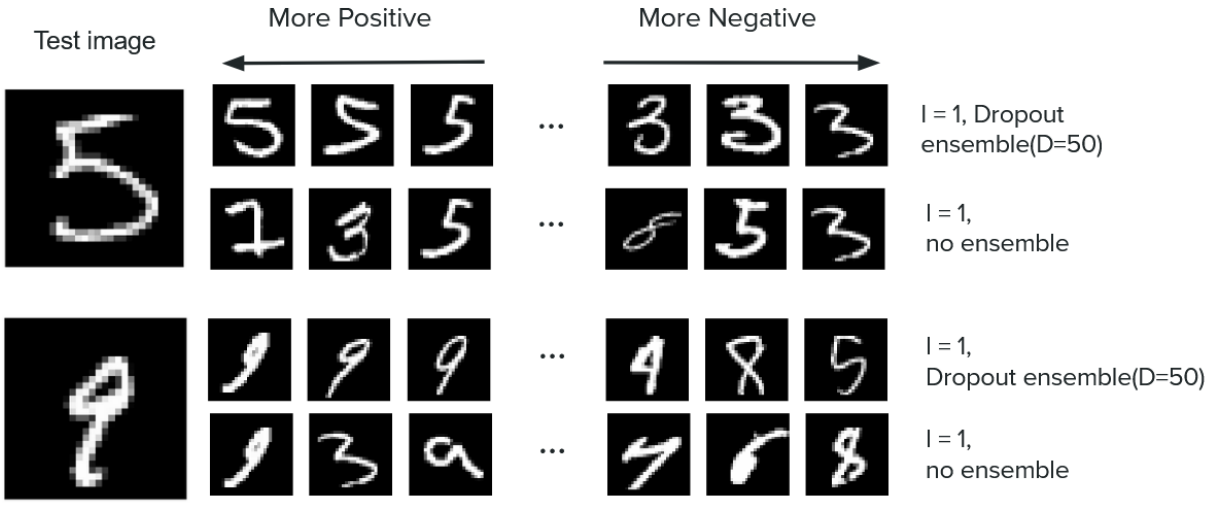


Figure 8: We present two randomly selected test images from MNIST-10 and the training samples that are most helpful (highest score) and most detracting (lowest score) calculated by TRAK with and without the Dropout Ensemble.

For Dropout Ensemble, we study the effect of dropout rate under the MNIST-10 + MLP setting. In the paper, we default to a dropout rate = 0.1. In these experiments, we examine the dropout rate range from 0.01 to 0.25. We can observe that the advantage of Dropout Ensemble persists. The results are presented in Table 14.

We study the effect of the LoRA rank/alpha and #fine-tuning epochs under one representative experimental configuration: Vision Transformer on the CIFAR-10 dataset. In the paper, our default parameters are: rank=8, alpha=8, and #fine-tuning epochs=30. In this analysis, we consider rank/alpha=32 and #fine-tuning epochs=60. Each hyperparameter is varied individually while the other is fixed. We can observe that the advantage of LoRA Ensemble persists across different settings. The results are presented in Table 15 and Table 16.

F Proofs

F.1 Proof of Lemma 5.1

Proof. We proceed by analyzing the bias and variance contributions for both the naive ensemble and the two-step ensemble. We first analyze the bias of the two estimators. Let $\mu_i = \mathbb{E}[\tau(\Theta^{(i)})]$ and $\mu_{(k,d)} = \mathbb{E}[\tau(\Theta^{(k,d)})]$ represent the expected values of the individual and variant estimators, respectively. Since we assume that each individual estimate $\tau(\Theta^{(i)})$ and $\tau(\Theta^{(k,d)})$ have the same and identical distribution, we have $\mu_i = \mu_{(k,d)}$,

Ensembling methods	D \ #ckpts	1	5
Naive Independent Ensemble	N/A	0.778	0.810
Dropout Ensemble	10	0.802	0.849
	25	0.810	0.849

Table 12: The AUC (Area-Under-Curve) of noisy label detection on MNIST-10 with 20% randomly flipped training samples. TRAK with efficient ensembles can outperform naive TRAK under all settings with drastically fewer computational cost.

Ensembling methods	D \ I	1	3	5
Naive Independent Ensemble	N/A	89.95	91.47	91.76
Dropout Ensemble	5	91.54	91.58	91.76
	10	91.54	91.67	91.76

Table 13: The test accuracy of noisy label detection on MNIST-10 with 10% randomly flipped training samples. TRAK with efficient ensembles can outperform naive TRAK under all settings with drastically fewer computational costs.

and thus

$$\text{Bias}^2(\tau_{\text{ens}}) = \left(\frac{1}{I}\sum_{i=1}^{I}\mu_i - \tau(\Theta^*)\right)^2 = \left(\frac{1}{KD}\sum_{k=1}^{K}\sum_{d=1}^{D}\mu_{(k,d)} - \tau(\Theta^*)\right)^2 = \text{Bias}^2(\tau_{\text{2-step}}) \tag{6}$$

Therefore,the difference in squared error Δ arises solely from the variance terms.

We then analyze the variance of the two estimators. For the naive ensemble estimator τ_{ens}, the variance is given by:

$$\text{Var}(\tau_{\text{ens}}) = \frac{1}{I^2}\sum_{i=1}^{I}\text{Var}(\tau(\Theta^{(i)})) + \frac{2}{I^2}\sum_{i<j}\text{Cov}(\tau(\Theta^{(i)}), \tau(\Theta^{(j)})) \tag{7}$$

Using the simplified notation, where $\Sigma_{1,1} = \text{Var}(\tau(\Theta^{(i)}))$ and $\Sigma_{i,j} = \text{Cov}(\tau(\Theta^{(i)}), \tau(\Theta^{(j)}))$ for $i \neq j$, this can be written as:

$$\text{Var}(\tau_{\text{ens}}) = \frac{1}{I}\left(\Sigma_{1,1} + (I-1)\Sigma_{i,j}\right) \tag{8}$$

For the two-step ensemble estimator $\tau_{\text{2-step}}$, the variance is given by:

$$\text{Var}(\tau_{\text{2-step}}) = \frac{1}{(KD)^2}\sum_{k=1}^{K}\sum_{d=1}^{D}\text{Var}(\tau(\Theta^{(k,d)})) + \frac{2}{(KD)^2}\sum_{(k,d)<(k',d')}\text{Cov}(\tau(\Theta^{(k,d)}), \tau(\Theta^{(k',d')})) \tag{9}$$

This variance can be decomposed into within-group and between-group covariances. Let $\Sigma_{(k,m),(k,n)} = \text{Cov}(\tau(\Theta^{(k,m)}), \tau(\Theta^{(k,n)}))$ represent the within-group covariance (i.e., between variants of the same base estimator), and let $\Sigma_{(k,m),(l,n)} = \text{Cov}(\tau(\Theta^{(k,m)}), \tau(\Theta^{(l,n)}))$ represents the between-group covariance (i.e., between variants of different base estimators). The variance simplifies to:

$$\text{Var}(\tau_{\text{2-step}}) = \frac{1}{KD}\left(\Sigma_{(k,m),(k,m)} + (D-1)\Sigma_{(k,m),(k,n)} + D(K-1)\Sigma_{(k,m),(l,n)}\right) \tag{10}$$

Dropout Rate	LDS
Naive	0.122
0.01	0.210
0.02	0.219
0.05	0.236
0.1	0.249
0.15	0.253
0.2	0.227
0.25	0.213

Table 14: The TDA efficacy using different dropout rate on MNIST-10 + MLP with $D = 10$ for DROPOUT ENSEMBLE.

	I=1	I=3
Naive	0.0278	0.0464
LoRA Ensemble (L=5)	0.0440	0.0511
LoRA Ensemble (L=10)	0.0515	0.0534

Table 15: The TDA efficacy using Lora rank = 32, lora alpha = 32, epoch = 30 for LoRA ENSEMBLE.

Finally, the difference in variance between the two estimators, denoted as Δ, is:

$$\Delta = \mathrm{Var}(\tau_{\text{ens}}) - \mathrm{Var}(\tau_{\text{2-step}}) \tag{11}$$

Substituting the variance expressions for τ_{ens} and $\tau_{\text{2-step}}$, we have:

$$\Delta = \frac{1}{I}\left(\Sigma_{1,1} + (I-1)\Sigma_{i,j}\right) - \frac{1}{KD}\left(\Sigma_{(k,m),(k,m)} + (D-1)\Sigma_{(k,m),(k,n)} + D(K-1)\Sigma_{(k,m),(l,n)}\right) \tag{12}$$

This expression shows that the difference in error depends on the number of estimators (I in the naive ensemble and KD in the two-step ensemble) and the covariance structure among the estimators. The two-step ensemble will outperform the naive ensemble if the within-group and between-group covariance are sufficiently small compared to the overall covariance in the naive ensemble. □

F.2 Derivation of the error difference for $K = I$ and $KD = I$

For both derivations, we use the following assumptions:

$$\Sigma_{1,1} = \Sigma_{(k,m),(k,m)} \quad \text{and} \quad \Sigma_{i,j} = \Sigma_{(k,m),(l,n)} \text{ for } i \neq j, k \neq l \tag{13}$$

This first equality implies that the variance of each estimator in both ensemble approaches is the same. The second equality implies that the covariance between different estimators in the naive ensemble is the same as the covariance between different estimator variants in the two-step ensemble. These are reasonable assumptions following the i.d. assumption on the individual estimators.

Case 1: $K = I$. Substitute $K = I$ into the expression for Δ and use the assumption $\Sigma_{1,1} = \Sigma_{(k,m),(k,m)}$ and $\Sigma_{i,j} = \Sigma_{(k,m),(l,n)}$, this simplifies Δ to:

	I=1	I=3
Naive	0.0278	0.0464
LoRA Ensemble (L=5)	0.0427	0.0525
LoRA Ensemble (L=10)	0.0507	0.0603

Table 16: The TDA efficacy using Lora rank = 8, lora alpha = 8, epoch = 60 for LoRA Ensemble.

$$\Delta = \frac{1}{I}\left(\Sigma_{1,1} + (I-1)\Sigma_{i,j}\right) - \frac{1}{ID}\left(\Sigma_{1,1} + (D-1)\Sigma_{(k,m),(k,n)} + D(I-1)\Sigma_{i,j}\right) \tag{14}$$

$$= \frac{1}{I}\Sigma_{1,1} + \frac{I-1}{I}\Sigma_{i,j} - \frac{1}{ID}\Sigma_{1,1} - \frac{D-1}{ID}\Sigma_{(k,m),(k,n)} - \frac{I-1}{I}\Sigma_{i,j} \tag{15}$$

$$= \frac{1}{I}\Sigma_{1,1} - \frac{1}{ID}\Sigma_{1,1} - \frac{D-1}{ID}\Sigma_{(k,m),(k,n)} \tag{16}$$

$$= \frac{D-1}{ID}\Sigma_{1,1} - \frac{D-1}{ID}\Sigma_{(k,m),(k,n)} \tag{17}$$

$$= \frac{D-1}{ID}\left(\Sigma_{1,1} - \Sigma_{(k,m),(k,n)}\right) \tag{18}$$

Case 2: $KD = I$**.** Substitute $KD = I$ into the expression for Δ and use the assumption $\Sigma_{1,1} = \Sigma_{(k,m),(k,m)}$ and $\Sigma_{i,j} = \Sigma_{(k,m),(l,n)}$, this simplifies Δ to:

$$\Delta = \frac{1}{I}\left(\Sigma_{1,1} + (I-1)\Sigma_{i,j}\right) - \frac{1}{I}\left(\Sigma_{1,1} + (D-1)\Sigma_{(k,m),(k,n)} + (I-D)\Sigma_{i,j}\right) \tag{19}$$

$$= \frac{1}{I}\Sigma_{1,1} + \frac{I-1}{I}\Sigma_{i,j} - \frac{1}{I}\Sigma_{1,1} - \frac{D-1}{I}\Sigma_{(k,m),(k,n)} - \frac{I-D}{I}\Sigma_{i,j} \tag{20}$$

$$= \frac{I-1}{I}\Sigma_{i,j} - \frac{D-1}{I}\Sigma_{(k,m),(k,n)} - \frac{I-D}{I}\Sigma_{i,j} \tag{21}$$

$$= \frac{D-1}{I}\Sigma_{i,j} - \frac{D-1}{I}\Sigma_{(k,m),(k,n)} \tag{22}$$

$$= \frac{D-1}{I}\left(\Sigma_{i,j} - \Sigma_{(k,m),(k,n)}\right) \tag{23}$$

G Discussion of Dropout Ensemble (forward-only)

Dropout Ensemble (forward-only) performs well in the experiments demonstrated in Figure 5, especially considering the competitive efficacy-efficiency trade-off. A potential explanation for this phenomenon is that the term (Q) affects the performance "substantially" as shown in TRAK (Park et al., 2023)'s ablation study. Focusing on mitigating the effect of randomness on this term and caching other terms could improve the TDA performance effectively and efficiently. It is worth noting that this phenomenon may be closely related to some open questions existing in the TRAK algorithm. For instance, empirical evidence indicates that a particular approach to ensembling models, known as "term-wise" ensemble in the TRAK paper, plays a critical role in TRAK's superior performance. However, the underlying reasons for this remain insufficiently understood.