TRAK: Attributing Model Behavior at Scale

Sung Min Park [*1] **Kristian Georgiev** [*1] **Andrew Ilyas** [*1] **Guillaume Leclerc** [1] **Aleksander Mądry** [1]

Abstract

The goal of *data attribution* is to trace model predictions back to training data. Despite a long line of work towards this goal, existing approaches to data attribution tend to force users to choose between computational tractability and efficacy. That is, computationally tractable methods can struggle with accurately attributing model predictions in non-convex settings (e.g., in the context of deep neural networks), while methods that are effective in such regimes require training thousands of models, which makes them impractical for large models or datasets. In this work, we introduce TRAK (Tracing with the Randomly-projected After Kernel), a data attribution method that is both effective *and* computationally tractable for large-scale, differentiable models. In particular, by leveraging only a handful of trained models, TRAK can match the performance of attribution methods that require training thousands of models. We demonstrate the utility of TRAK across various modalities and scales: image classifiers trained on ImageNet, vision-language models (CLIP), and language models (BERT and mT5). We provide code for using TRAK (and reproducing our work) at https://github.com/MadryLab/trak.

1. Introduction

Training data is a key driver of model behavior in modern machine learning systems. Indeed, model errors, biases, and capabilities can all stem from the training data (Ilyas et al., 2019; Gu et al., 2017; Geirhos et al., 2019). Furthermore, improving the quality of training data generally improves the performance of the resulting models (Huh et al., 2016; Lee et al., 2022). The importance of training data to model behavior has motivated extensive work on *data attribution*, i.e., the task of tracing model predictions back to the training examples that informed these predictions. Recent work demonstrates, in particular, the utility of data attribution methods in applications such as explaining predictions (Koh & Liang, 2017; Ilyas et al., 2022), debugging model behavior (Kong et al., 2022), assigning data valuations (Ghorbani & Zou, 2019), detecting poisoned or mislabeled data (Lin et al., 2022; Hammoudeh & Lowd, 2022a), and curating data (Khanna et al., 2019; Liu et al., 2021; Jia et al., 2021).

However, a recurring tradeoff in the space of data attribution methods is that of *computational demand* versus *efficacy*. On the one hand, methods such as influence approximation (Koh & Liang, 2017; Schioppa et al., 2022) or gradient agreement scoring (Pruthi et al., 2020) are computationally attractive but can be unreliable in non-convex settings (Basu et al., 2021; Ilyas et al., 2022). On the other hand, sampling-based methods such as empirical influence functions (Feldman & Zhang, 2020), Shapley value estimators (Ghorbani & Zou, 2019) or datamodels (Ilyas et al., 2022) are more successful at accurately attributing predictions to training data but require training thousands (or tens of thousands) of models to be effective. We thus ask:

Are there data attribution methods that are both scalable and effective in large-scale non-convex settings?

In the remainder of this paper, we answer this question in the affirmative. Specifically:

- Building on ideas from prior work (Ilyas et al., 2022), we begin by providing a unifying metric for data attribution methods. We examine existing data attribution methods and find that current approaches are either prohibitively expensive, or cheap but ineffective.
- Leveraging results from classical statistics (Pregibon, 1981; Johnson & Lindenstrauss, 1984), and a connection between differentiable models and their corresponding kernel machines (Jacot et al., 2018), we derive a new data attribution method called TRAK.
- On standard computer vision and NLP tasks (CIFAR, ImageNet, QNLI), TRAK scores are 100-1000x faster than comparably effective methods (see Figure 1).

[*]Equal contribution [1]Department of EECS, Massachusetts Institute of Technology, Cambridge, MA. Correspondence to: Sung Min Park <sp765@mit.edu>, Kristian Georgiev <krisgrg@mit.edu>, Andrew Ilyas <ailyas@mit.edu>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

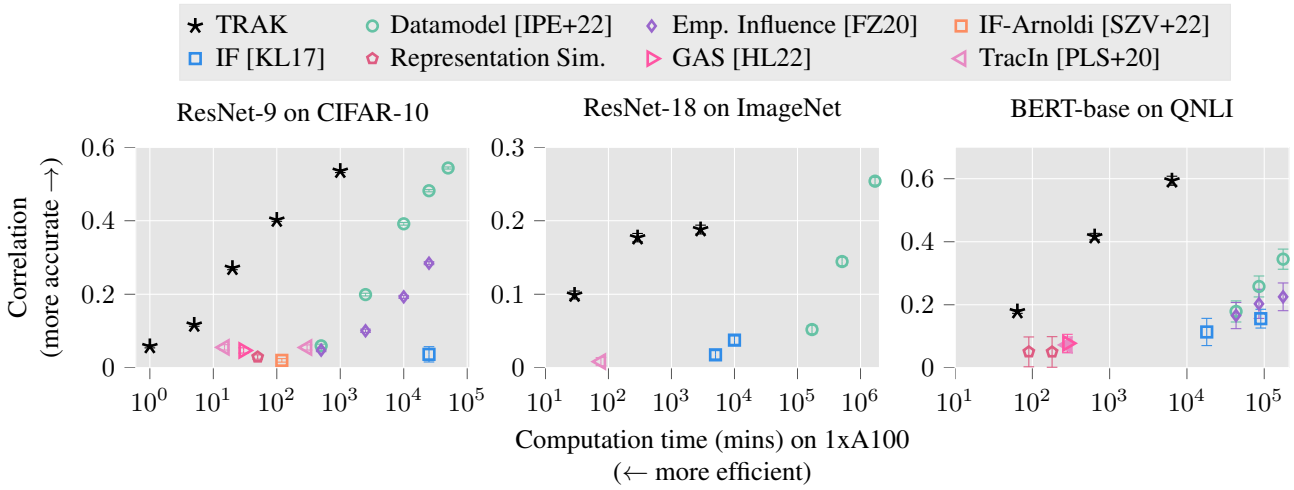

Figure 1. Our data attribution method TRAK achieves state-of-the-art tradeoffs between speed and efficacy. Here, we benchmark its performance relative to prior methods on ResNet-9 models trained on CIFAR-10, ResNet-18 models trained on ImageNet, and BERT-BASE models finetuend on QNLI. The x-axis indicates the time (in minutes) it takes to run each method on a single A100 GPU (see Appendix C.5 for details). The y-axis indicates the method's efficacy as measured by its ability to make accurate counterfactual predictions (see Definition 2.2 for the precise metric); error bars indicate 95% bootstrap confidence intervals.

- We can thus scale TRAK to models and tasks that were previously untenable for data attribution methods—we study CLIP models and a 300M-parameter language model mT5.
- We provide an easy-to-use PyTorch API for TRAK, allowing the community to apply data attribution to other large-scale models.

2. Motivation and Setup

We begin with a focus on the supervised learning regime. We will denote by $S = \{z_1, \ldots, z_n\}$ an ordered training set of examples, where each $z_i = (x_i, y_i) \in \mathcal{Z}$ is an input-label pair. We represent machine learning models (implicitly) using a *model output function* $f(z; \theta)$, which maps an example of interest z and model parameters θ to a real number. There are a variety of model output functions that one can employ—for example, the loss $L(z; \theta)$ of the model on the example z is a natural choice. Ultimately, though, the appropriate model output function to use will depend on the setting that we are studying.

Throughout this work, we also assume that models are trained to minimize the empirical training loss, i.e., that the parameters of these models are given by

$$\theta^\star(S) := \arg\min_\theta \sum_{z_i \in S} L(z_i; \theta), \tag{1}$$

where, again, $L(z_i; \theta)$ is the model training loss on example z_i. We write $\theta^\star$ as a function of S as we will later consider varying S—but when S is clear from the context, we omit it and just write $\theta^\star$.

In this paper, our overarching goal is to trace model predictions back to the composition of training data. This goal—which we refer to as *data attribution*—is not new. Prior work has approached it using methods such as influence functions and their many variants (Hampel et al., 2011; Koh & Liang, 2017); sampling-based estimators such as Shapley values (Lundberg & Lee, 2017), empirical influences (Feldman & Zhang, 2020), and datamodels (Ilyas et al., 2022); as well as various other approaches (Yeh et al., 2018; Pruthi et al., 2020; Hammoudeh & Lowd, 2022b). Each of these methods implements a similar interface: given a model and an output of interest (e.g., loss for a given prediction), a data attribution method computes a *score* for each training input indicating its importance to the output of interest. Definition 2.1 below makes this interface precise:

Definition 2.1 (*Data attribution*)**.** Consider an ordered training set of examples $S = \{z_1, \ldots, z_n\}$ and a model output function $f(z; \theta)$. A *data attribution method* $\tau(z, S)$ is a function $\tau : \mathcal{Z} \times \mathcal{Z}^n \to \mathbb{R}^n$ that, for any example $z \in \mathcal{Z}$ and a training set S, assigns a (real-valued) score to each training input $z_i \in S$ indicating its importance[1] to the model output $f(z; \theta^\star(S))$. When the second argument S is clear from the context, we will omit the second argument and simply write $\tau(z)$.

Evaluating attribution methods. Given the variety of existing data attribution methods, we need a method to evaluate them in a consistent way. One popular approach is to simply manually inspect the training examples that the method identifies as most important for a given prediction or set of

[1]We make "importance" more precise in Definition 2.2.

predictions. Such manual inspection can be a useful sanity check, but is also often subjective and unreliable. A more objective alternative is to treat the scores from a data attribution method as estimates of some ground-truth parameters—such as leave-one-out influences (Koh & Liang, 2017; Basu et al., 2021) or Shapley values (Lundberg & Lee, 2017)—and then measure the accuracy of these estimates. However, getting access to these ground-truth parameters can be prohibitively expensive in large-scale settings. Finally, yet another possibility is to measure the utility of data attribution scores for an auxiliary task such as identifying mislabeled data (Koh & Liang, 2017; Hammoudeh & Lowd, 2022a) or active learning (Jia et al., 2021). This approach can indeed be a useful proxy for evaluating attribution methods, but the resulting metrics may be too sensitive to the particulars of the auxiliary task, making comparisons across different problems and settings difficult.

2.1. The linear datamodeling score (LDS)

Motivated by the above shortcomings of existing methodologies, we propose a new metric for evaluating data attribution methods. At the heart of our metric is the perspective that an effective data attribution method should be able to make accurate *counterfactual predictions* about how model outputs change when the training set is modified.

Inspired by Ilyas et al. (2022), we cast this counterfactual estimation task as that of predicting the model output function $f(z; \theta^\star(S'))$ corresponding to different subsets of the training set S'. More precisely, consider—for a fixed example of interest $z \in \mathcal{Z}$—the model output $f(z; \theta^\star(S'))$ arising from training on a subset $S' \subset S$ of the training set S (see (1)).[2] Since z is fixed and the learning algorithm $\theta^\star(\cdot)$ is fixed, we can view this model output as a function of S' alone. A good data attribution method should help us predict the former from the latter.

To operationalize this idea, we first need a way of converting a given data attribution method $\tau(\cdot)$ into a counterfactual predictor. Observing that the vast majority of data attribution methods are *additive*,[3] we define an attribution method's *prediction* of the model output for a subset $S' \subset S$ as the sum of the corresponding scores:

Definition 2.2 (*Attribution-based output predictions*)**.** Consider a training set S, a model output function $f(z; \theta)$, and a corresponding data attribution method τ (see Definition 2.1). The *attribution-based output prediction* of the model output $f(z; \theta^\star(S'))$ is defined as

$$g_\tau(z, S'; S) \coloneqq \sum_{i \,:\, z_i \in S'} \tau(z, S)_i = \tau(z, S) \cdot \mathbf{1}_{S'}, \quad (2)$$

where $\mathbf{1}_{S'}$ is the *indicator vector* of the subset S' of S (i.e., $(\mathbf{1}_{S'})_i = \mathbf{1}\{z_i \in S'\}$).

Intuitively, Definition 2.2 turns any data attribution method into a counterfactual predictor. Specifically, for a given counterfactual training set $S' \subset S$, the attribution method's prediction is simply the sum of the scores of the examples contained in S'.

Now that we have defined how to derive predictions from an attribution method, we can evaluate these predictions using the *linear datamodeling score*, defined as follows:

Definition 2.3 (*Linear datamodeling score*)**.** Consider a training set S, a model output function $f(z; \theta)$, and a corresponding data attribution method τ (see Definition 2.1). Let $\{S_1, \ldots, S_m : S_i \subset S\}$ be m randomly sampled subsets of the training set S, each of size $\alpha \cdot n$ for some $\alpha \in (0, 1)$. The *linear datamodeling score* (LDS) of a data attribution τ for a specific example $z \in \mathcal{Z}$ is given by

$$LDS(\tau, z) \coloneqq \boldsymbol{\rho}(\{f(z; \theta^\star(S_j)), g_\tau(z, S_j; S) : j \in [m]\}),$$

where $\boldsymbol{\rho}$ denotes Spearman rank correlation (Spearman, 1904). The attribution method's LDS for an entire test set is then simply the average per-example score.

The LDS is quantitative, simple to compute, and not tied to a specific task or modality.

2.2. An oracle for data attribution

Definition 2.3 immediately suggests an "optimal" approach to data attribution (at least, in terms of optimizing LDS). This approach simply samples random subsets $\{S_1, \ldots S_m\}$ of the training set; trains a model on each subset (yielding $\{\theta^\star(S_1), \ldots, \theta^\star(S_m)\}$); evaluates each corresponding model output function $f(z; \theta^\star(S_j))$; and then *fits* scores $\tau(z)$ that predict $f(z; \theta^\star(S_i))$ from the indicator vector $\mathbf{1}_{S_i}$ using (regularized) empirical risk minimization. Indeed, Ilyas et al. (2022) take exactly this approach—the resulting datamodel-based attribution for an example z is then

$$\tau_{\text{DM}}(z) \coloneqq \min_{\beta \in \mathbb{R}^n} \sum_{i=1}^{m} \left(\beta^\top \mathbf{1}_{S_i} - f(z; \theta^\star(S_i))\right)^2 + \lambda \|\beta\|_1. \quad (3)$$

The attributions $\tau_{\text{DM}}(z)$ yield counterfactual predictions that are highly correlated with true model outputs (see Figure 1). Unfortunately, however, estimating accurate linear predictors (3) requires thousands of samples $(S_j, f(z; \theta^\star(S_j)))$. Since each sample involves training a model from scratch, this direct estimator can be expensive to compute in large-scale settings. More generally, this limitation applies to all

[2] In many settings, the non-determinism of training makes this model output function a random variable, but we treat it as deterministic to simplify our notation. We handle non-determinism explicitly in Section 3.2.

[3] That is, they define the importance of a group of training examples to be the sum of the importances of the examples in the group. See Appendix E.6 for a more detailed discussion.

sampling-based attribution methods, such as empirical influences (Feldman & Zhang, 2020; Carlini et al., 2022) and Shapley values (Ghorbani & Zou, 2019; Jia et al., 2019).

In light of the above, we can view the approach of Ilyas et al. (2022) as an "oracle" of sorts—it makes accurate counterfactual predictions (and as a result has found downstream utility (Ilyas et al., 2022; Shah et al., 2022; Chang & Jia, 2022)), but is (often prohibitively) costly to compute.

2.3. Data attribution methods beyond sampling

We consider other existing data attribution methods—specifically, the ones that use only one (or a few) trained models—and evaluate them on our LDS benchmark.

Simulating re-training with influence functions. The bottleneck of the "oracle" datamodels attribution method (3) from above is that obtaining each sample $(S_j, f(z; S_j))$ requires re-training our model from scratch on each subset S_j. An alternative approach could be to *simulate* the effect of this re-training by making some structural assumptions about the model being studied—e.g., that its loss is locally well-approximated by a quadratic. This idea has inspired a long line of work around *influence function estimation* (Koh & Liang, 2017; Pruthi et al., 2020; Schioppa et al., 2022). The resulting *influence function attributions* accurately approximate linear models and other simple models, but can perform poorly in non-convex settings (e.g., in the context of deep neural networks) (Basu et al., 2021; Ilyas et al., 2022; Bae et al., 2022). Indeed, as we show in Figure 1, estimators based on influence functions significantly underperform on our LDS benchmark when evaluated on neural networks on standard vision and natural language tasks.

Heuristic measures of example importance. Other approaches use more heuristic measures of training example importance for data attribution. These include methods based on, e.g., representation space similarity (Zhang et al., 2018; Hanawa et al., 2021) or gradient agreement (Hammoudeh & Lowd, 2022a). While such methods often yield qualitatively compelling results, our experiments (again, see Figure 1) indicate that, similarly to influence-based estimators, they are unable to make meaningful counterfactual predictions about model outputs in the large-scale, non-convex settings we evaluate them on.

3. TRAK: Tracing with the Randomly-Projected After Kernel

We now present TRAK, a new data attribution method which is designed to be both effective and scalable in large-scale differentiable settings.

As a warm-up, and to illustrate the core primitive behind TRAK, we first study the simple case of logistic regression (Section 3.1). In this setting, data attribution is well-understood—in particular, there is a canonical attribution method (Pregibon, 1981) that is both easy-to-compute and highly effective (Wojnowicz et al., 2016; Koh et al., 2019). In Section 3.2, using this canonical attribution method as a primitive, we derive our data attribution method TRAK, which operates by reducing complex models back to the logistic regression case.[4]

3.1. Warmup: Data attribution for logistic regression

Consider the case where the model being studied is (a generalized form of) binary logistic regression. In particular, we consider a training set of n examples

$$S = \{z_1, \ldots, z_n : z_i = (x_i \in \mathbb{R}^d,\, b_i \in \mathbb{R},\, y_i \in \{-1, 1\})\},$$

where each example comprises an input $x_i \in \mathbb{R}^d$, a bias $b_i \in \mathbb{R}$, and a label $y_i \in \{-1, 1\}$. The final model parameters $\theta^\star(S)$ then minimize the log-loss over the training set, i.e.,

$$\theta^\star(S) := \arg\min_\theta \sum_{(x_i, y_i) \in S} -\log\left(\sigma\left(y_i \cdot (\theta^\top x_i + b_i)\right)\right), \tag{4}$$

where $\sigma(\cdot)$ is the sigmoid function. (Note that $b_i = 0$ correspond to ordinary logistic regression.) The natural choice of *model output function* in this case is then the "raw logit" function:

$$f(z; \theta) := \theta^\top x + b, \qquad \text{where } z = (x, b, y). \tag{5}$$

Data attribution in this simple setting is a well-studied problem. In particular, the *one-step Newton approximation* (Pregibon, 1981; Wojnowicz et al., 2016; Rad & Maleki, 2018; Koh et al., 2019), which we present as a data attribution method τ_{NS} below, is a standard tool for analyzing and understanding logistic regression models in terms of their training data. (We present the theoretical basis for this method in Appendix E.1.)

Definition 3.1 (One-step Newton approximation (Pregibon, 1981))**.** For logistic regression, we define the Newton step data attribution method τ_{NS} as the approximate leave-one-out influence (Pregibon, 1981) of training examples $z_i = (x_i, b_i, y_i)$ on the model output function (5). That is,

$$\tau_{\text{NS}}(z)_i := \frac{x^\top (X^\top R X)^{-1} x_i}{1 - x_i^\top (X^\top R X)^{-1} x_i \cdot p_i^\star (1 - p_i^\star)} (1 - p_i^\star) \tag{6}$$

$$\approx f(z; \theta^\star(S)) - f(z; \theta^\star(S \setminus z_i)) \tag{7}$$

where $X \in \mathbb{R}^{n \times k}$ is the matrix of stacked inputs x_i, $p_i^\star := (1 + \exp(-y_i \cdot f(z_i; \theta^\star)))^{-1}$ is the predicted correct-class probability at $\theta^\star$ and R is a diagonal $n \times n$ matrix with $R_{ii} = p_i^\star (1 - p_i^\star)$.

[4] Note that we focus on logistic regression for simplicity—more generally one can adapt TRAK to any setting where the training loss is convex in the model output; see Appendix E.1.

If our model class of interest was binary logistic regression, we could simply apply Definition 3.1 to perform data attribution. As we discuss, however, our goal is precisely to scale data attribution *beyond* such convex settings. To this end, we next derive our data attribution method TRAK (Tracing with the Randomly-projected After Kernel) which leverages τ_{NS} (Definition 3.1) as a primitive.

3.2. TRAK for binary (non-linear) classifiers

We now present our method (TRAK) for scaling data attribution to non-convex differentiable settings. The key primitive here will be Definition 3.1 from above—in particular, we will show how to adapt our problem into one to which we can apply the approximation (7).

For ease of exposition, we will first show how to compute τ_{TRAK} in the context of binary classifiers trained with the negative log-likelihood loss. (We later generalize TRAK to other types of models, e.g., to multi-class classifiers in Appendix E.5, to contrastive models in Section 5.1, and to language models in Section 5.2.) In this setting, let the model output function $f(z;\theta)$ be the raw output (i.e., the logit) of a binary classifier with parameters θ.[5] The final parameters of the model can thus be written as

$$\theta^\star(S) = \arg\min_\theta \sum_{(x_i,y_i)\in S} -\log\left[\sigma\left((-y_i \cdot f(z_i;\theta))\right)\right]. \tag{8}$$

Note that unlike in Section 3.1, we do not assume that the model itself is linear—e.g., the model might be a deep neural network parameterized by weights θ.

We implement TRAK as a sequence of five steps: linearization, dimensionality reduction, one-step newton approximation, ensembling, and soft thresholding. We discuss these steps in more depth below.

(Step 1) Linearizing the model. Recall that our goal here is to apply the data attribution method τ_{NS} from Definition 3.1. The main roadblock to applying Definition 3.1 in our setting is that we are studying a *non-linear* model—that is, our model output function may not be a linear function of θ. We address this issue by approximating $f(z;\theta)$ with its Taylor expansion centered around the final model parameters $\theta^\star$:

$$\hat{f}(z;\theta) \coloneqq f(z;\theta^\star) + \nabla_\theta f(z;\theta^\star)^\top(\theta - \theta^\star). \tag{9}$$

This approximation suggests a change in perspective—rather than viewing $f(z;\theta)$ as a non-linear model acting on inputs x, we can view it as a *linear* model acting on inputs $\nabla_\theta f(z;\theta^\star)$. In particular, rewriting the loss minimization (8) by replacing $f(z;\theta)$ with $\hat{f}(z;\theta)$ and defining the variables $g_i \coloneqq \nabla_\theta f(z_i;\theta^\star)$ and $b_i \coloneqq f(z_i;\theta^\star) - \nabla_\theta f(z_i;\theta^\star)^\top\theta^\star$ yield

$$\theta^\star(S) = \arg\min_\theta \sum_{(g_i,b_i,y_i)} -\log\left[\sigma\left(y_i \cdot \left(\theta^\top g_i + b_i\right)\right)\right]. \tag{10}$$

Comparing (10) to (4) (from Section 3.1) makes it clear that we can view $\theta^\star$ as the solution to a (generalized) logistic regression, in which the inputs x_i are model gradients g_i, the bias terms are b_i and the labels y_i remain the same.

Note: In the context of neural networks, we can view Step 1 as replacing the binary classifier with its empirical neural tangent kernel (eNTK) approximation (Jacot et al., 2018; Atanasov et al., 2022; Wei et al., 2022). We discuss how TRAK connects to the eNTK in more detail in Appendix B.

(Step 2) Reducing dimensionality with random projections. The linear approximation from Step 1 dramatically simplifies our model class of interest from a highly non-linear classifier to simple logistic regression. Still, the resulting logistic regression can be extremely high dimensional: the input dimension of the linear model (9) is the number of parameters of the original model (which can be on the order of millions), not the dimensionality of the inputs x_i.

To reduce the dimensionality of this problem, we leverage a classic result of Johnson & Lindenstrauss (1984). This result guarantees that multiplying each gradient $g_i = \nabla_\theta f(z_i;\theta^\star) \in \mathbb{R}^p$ by a random matrix $\mathbf{P} \sim \mathcal{N}(0,1)^{p\times k}$ for $k \ll p$ preserves inner products $g_i^\top g_j$ with high probability[6] (while significantly reducing the dimension). Thus, we define the "feature map" $\phi : \mathcal{Z} \to \mathbb{R}^k$ as

$$\phi(z) \coloneqq \mathbf{P}^\top \nabla_\theta f(z;\theta^\star), \tag{11}$$

i.e., a function taking an example z to its corresponding projected gradient, and from now on replace g_i with

$$\phi_i \coloneqq \phi(z_i) = \mathbf{P}^\top g_i = \mathbf{P}^\top \nabla_\theta f(z_i;\theta^\star). \tag{12}$$

(Step 3) Estimating influences. Now that we have simplified our model of interest to a logistic regression problem of tractable dimension, we can finally adapt Definition 3.1.

To this end, recall that the training "inputs" are now the (projected) gradients ϕ_i (see (12)). We thus replace the matrix X in (7) with the matrix $\Phi \coloneqq [\phi_1;\ldots,\phi_n] \in \mathbb{R}^{n\times k}$ of stacked projected gradients. We also find empirically that both the denominator in (7) and the diagonal matrix R have little effect on the resulting estimates, and so we omit them from our adapted estimator. Our estimator for attribution

[5] Note that for the special case of binary classifiers, the model output function that we define (i.e., $f(z;\theta) = f((x,y);\theta)$) depends only on the input x, and not on the label y. When we generalize TRAK to more complex losses (e.g., Appendix E.5) the model output function will involve both x and y.

[6] In Appendix E.2 we discuss why preserving inner products suffices to preserve the structure of the logistic regression.

scores for an example of interest z thus becomes:

$$\tau(z, S) := \phi(z)^\top (\Phi^\top \Phi)^{-1} \Phi^\top \mathbf{Q}, \tag{13}$$

where we recall from (11) that $\phi(z) = \mathbf{P}^\top \nabla_\theta f(z; \theta^\star)$, and where we define

$$\mathbf{Q} := \text{diag}(\{1 - p_i^\star\}) = \text{diag}\left(\{1 - \sigma\left(y_i \cdot f(z_i; \theta^\star)\right)\}\right) \tag{14}$$

as the $n \times n$ diagonal matrix of "one minus correct-class probability" terms.

(Step 4) Ensembling over independently trained models. So far, our analysis ignores the fact that in many modern settings, training is non-deterministic: applying the same learning algorithm to the same training dataset (i.e., varying only the random seed) can yield models with (often significantly) differing behavior (Nguyen et al., 2021b; D'Amour et al., 2020). Non-determinism poses a problem for data attribution because by definition, we cannot explain such seed-based differences in terms of the training data.

To reduce the impact of such seed-based differences, we aggregate the estimator (13) across an ensemble of models trained on different randomly selected subsets $S_1, \ldots, S_M \subset S$, leading to the final estimator τ_M:

$$\frac{1}{M} \sum_{i=1}^{M} \mathbf{Q}_m \cdot \left(\frac{1}{M} \sum_{i=1}^{M} \phi_m(z)^\top (\Phi_m^\top \Phi_m)^{-1} \Phi_m^\top \right) \tag{15}$$

where Φ_m are the corresponding projected gradients from each model $\theta^\star(S_m)$; $\phi_m(z)$ is the featurized example z under model $\theta^\star(S_m)$; and $\mathbf{Q}_m$ is the corresponding matrix of probabilities as defined in Equation (14).[7]

(Step 5) Inducing sparsity via soft-thresholding. Ilyas et al. (2022) find that for neural networks attribution scores are often sparse—that is, each test example depends on only a few examples from the training set. Motivated by this observation, we post-process the attribution scores from Step 4 via *soft thresholding*. After applying the soft-thresholding operator $\mathcal{S}$, our final estimator becomes

$$\tau_{\text{TRAK}}(z, S) := \mathcal{S}\left(\tau_M(z, S); \widehat{\lambda}\right) \tag{16}$$

where threshold $\widehat{\lambda}$ is selected via cross-validation (see Appendix E.4 for more details).

3.3. Implementing TRAK

We summarize our final algorithm for computing the data attribution method τ_{TRAK} in the general multi-class case in Algorithm 1. To make Algorithm 1 efficient even for large models, we implemented a highly optimized random projector, which we discuss in Appendix D.

[7] We motivate the use of random subsets in Appendix E.3 and our method of averaging in Appendix G.

4. Evaluating TRAK

We evaluate TRAK in a variety of vision and NLP settings. To this end, we compare TRAK with existing data attribution methods and show that it achieves significantly better tradeoffs between efficacy and computational efficiency.

4.1. Experimental setup

We use ResNet-9 classifiers trained on CIFAR-10;ResNet-18 classifiers trained on the 1000-class ImageNet dataset, and pre-trained BERT models finetuned on the QNLI (Question-answering Natural Language Inference) task from the GLUE benchmark (Wang et al., 2018). (See Appendix C.1 for more details.)

To put TRAK's performance into context, we also evaluate a variety of existing attribution methods (see Appendix C.3 for details). For each method and each dataset we consider, we compute its linear datamodeling score (LDS) as described in Definition 2.3; see Appendix C.4 for more details.

We quantify the computational cost of each attribution method using two metrics: the *total wall time* of computing attribution scores on a single A100 GPU, and the *total number of trained models used*. The latter is implementation-independent, while the former is more interpretable and factors in expensive operations such as matrix inversion. We find that both metrics lead to similar conclusions.

4.2. Results

Across all models and datasets that we consider, TRAK attains a significantly better tradeoff between efficacy (as measured by the LDS) and computational efficiency compared to other attribution methods (see Figures 1 and 8 and Table 2). Indeed, TRAK attains efficacy comparable to datamodels (which achieves the best performance among existing methods when unconstrained) with a computational footprint that is (on average) over 100x smaller.

TRAK-identified examples. In Figures 2 and 10 we show, for randomly chosen test examples from QNLI, CIFAR-10, and ImageNet, the training examples corresponding to the most positive and negative TRAK scores.

Understanding the roots of TRAK's performance. In Appendix G, we study the roots of TRAK's performance through an extensive ablation study. We vary the size of the ensemble, how we linearize the model of interest (Step 1 in Section 3.2), the dimension k of the random projection we use (Step 2 in Section 3.2), how we apply the Newton step attribution from Definition 3.1 (Step 3 in Section 3.2), and how we aggregate information from independently trained models (Step 4 in Section 3.2).

As a byproduct of this investigation, we find two ways of

Figure 2. We show a randomly selected test example and its corresponding most helpful (highest-scoring) and most detracting (lowest-scoring) training examples as identified by TRAK for ResNet-18 classifiers trained on ImageNet (bottom). We observe that TRAK-identified training examples are semantically similar to the target example, and that the helpful (detracting) examples are of the same (different) class as the target. See https://trak.csail.mit.edu for more examples.

computing TRAK at even lower cost: (a) leveraging models that have not been trained to convergence, and (b) using multiple checkpoints from the same model. Both of these optimizations dramatically reduce TRAK's computational cost without significantly degrading performance.

5. Applications of TRAK

We now illustrate the usefulness of TRAK through three additional applications: attributing CLIP models (Section 5.1), fact tracing language models (Section 5.2), and accelerating datamodel applications (Appendix A).

5.1. Attributing CLIP models

CLIP (Contrastive Language-Image Pre-training) (Radford et al., 2021) representations have become a versatile primitive bridging visual and language domains and is used, for example, for zero-shot classification (Radford et al., 2021) and as text encoders for latent diffusion models (Rombach et al., 2022). While the quality of these representations—as measured by aggregate metrics such as downstream zero-shot accuracy—appears to be driven largely by the properties and scale of the training datasets (Fang et al., 2022; Santurkar et al., 2022; Cherti et al., 2022), we lack a fine-grained understanding of how the composition of the training data contributes to learning well-aligned representations. To that end, we use TRAK to investigate how training data influences the resulting CLIP embeddings at a *local* level. That is, we want to pin-point training examples that cause a model to learn a *specific* image-caption pair association.

5.1.1. Computing TRAK for CLIP

Similarly to the classification setting we were considering so far, we need to choose an appropriate model output function (see, e.g., Equation (28)) to compute attribution scores with TRAK. Our choice (described in Appendix E.7) is motivated by the CLIP training loss and allows us to reduce this setting back to the classification case. We can then compute TRAK scores following the same approach as before.

5.1.2. Results

We train image-text models using the CLIP objective on MS COCO (Lin et al., 2014). We compare TRAK with TracIn and CLIP similarity distance baselines.

Visual analysis. Figure 3 displays a target example along with the corresponding most important training examples. The most helpful TRAK examples are the ones for which the captions contain the phrase "a couple of animals" but where the images do not necessarily feature giraffes (possibly because the target caption does not mention "giraffe" either). On the other hand, the most helpful examples according to CLIP similarity distance all feature giraffes. These differences suggest that TRAK attribution scores may capture significantly different traits from CLIP similarity distance.

Counterfactual evaluation. We next investigate to what extent training examples identified by each attribution method affect the CLIP model's ability to learn a given image-caption association. Specifically, we say that a CLIP model has *learned* a given association between an image and a caption whenever their corresponding image and caption embeddings have high cosine similarity relative to other image-caption pairs. To evaluate each attribution method, for a given target image-caption pair, we remove from the training set the k examples with the most positive attribution scores, and then re-train ten models from scratch. Finally, we measure the average decrease in cosine similarity between the embeddings of target image and caption pair, and average this result over different target pairs.

Our results (Figure 4) indicate that removing training inputs identified by TRAK can significantly degrade the model's ability to learn the target image-caption pair, while using CLIP or TracIn is less effective.

5.2. Fact tracing for large language models (mT5)

As large language models are deployed in a variety of contexts, there is an emerging need to be able to attribute models' outputs back to specific data sources (Bohnet et al.,

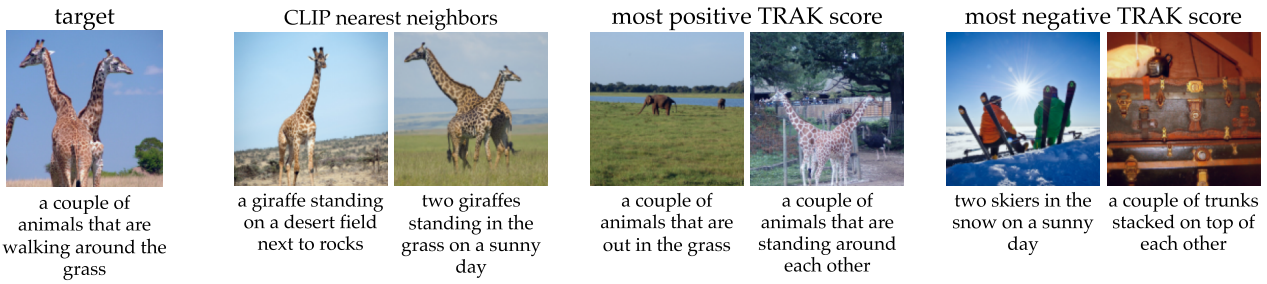

Figure 3. Attributing CLIP trained on MS COCO. The first column shows two target image-caption pairs from the validation set of MS COCO. The second two columns display the nearest neighbors to the target in CLIP embedding space (using the average of image and text cosine similarities). The next two columns show the train set samples that, according to TRAK, are most helpful for aligning the image embedding to the caption embedding. Similarly, the last two columns display the train samples that are the most detracting from aligning the image and caption embeddings. In Appendix F.2, we display more examples and also compare to TracIn.

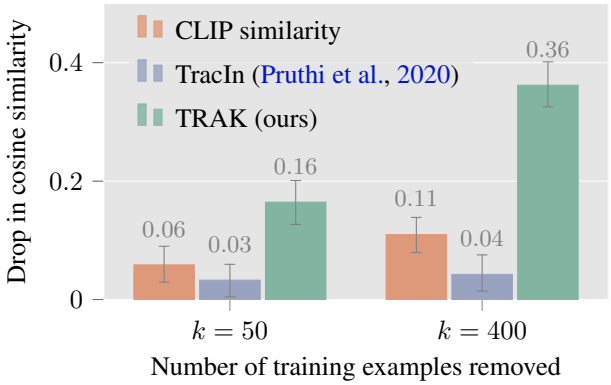

Figure 4. Counterfactually evaluating CLIP attributions. We measure how the cosine similarity between target image and caption embeddings is affected when we re-train a CLIP model after removing the most influential training examples—as identified by TRAK, TracIn, and CLIP similarity distance. We report the *decrease* in cosine similarity, averaged over 100 randomly selected image-caption pairs from the validation set. Error bars represent 95% confidence intervals.

2022). To that end, we study *fact tracing* (Akyurek et al., 2022), i.e., the task of identifying the training examples that cause a language model to generate a given "fact."

A benchmark for fact tracing. Akyurek et al. (2022) develop a testbed for the fact tracing problem by way of a dataset called FTRACE-TREX. The dataset consists of a set of "abstracts" and a set of "queries," both of which pertain to the same database of "facts." Each abstract is annotated with a set of facts it expresses, and each query with the (single) fact that it asks about. As a part of the task setup, one finetunes a pre-trained language model on the set of abstracts using *masked language modeling*,[8] and then evaluates this model's correctness on each query in the query set. This step defines a set of "novel facts," i.e., queries that the model answers correctly *only after* finetuning.

Akyurek et al. (2022) reason that each novel fact (as identified above) should have been learned (during finetuning) from the abstracts that express the same fact. The benchmark thus evaluates a given data attribution method's ability to retrieve, for each novel fact, the abstracts in the training set that express the same fact (called the *ground-truth proponents* of the query.)

In particular, observe that applying a data attribution method $\tau(\cdot)$ to a particular query (treating the set of abstracts as the training set) yields scores that we can use as a ranking over the set of the abstracts. Akyurek et al. (2022) compute the *mean reciprocal rank* (MRR) of the ground-truth proponents in this ranking (see Appendix H.1), a standard metric from information retrieval, to quantify the efficacy of $\tau(\cdot)$ at fact tracing. We evaluate TRAK on this benchmark, along with two baselines from (Akyurek et al., 2022), TracIn (Pruthi et al., 2020) and the information retrieval method BM25.

Computing TRAK scores for language models. To apply TRAK to this setting, we again need to choose an appropriate model output function. Observing that we can interpret that masked language modeling objective as a sequence of v-way classification problems over the masked tokens (where v is the vocabulary size), we choose the model output function for this setting to be the sum of the "canonical" model output function (28) for each of the v-way classification problems (see Appendix H.3 for more details).

5.2.1. RESULTS AND DISCUSSION

We find that while TRAK significantly outperforms TracIn on the FTRACE-TREX benchmark (0.42 vs. 0.09 MRR), neither method matches the performance of the information

[8] In masked language modeling (Raffel et al., 2020), the language model is asked to predict the tokens corresponding to a masked-out portion of the input. In FTRACE-TREX, either a subject or object in the abstract is masked out.

retrieval baseline BM25 (0.77 MRR).

To understand the possible roots of TRAK's underperformance relative to BM25 on FTRACE-TREX, we carry out a counterfactual analysis. Specifically, for a subset $S^\star$ of the FTRACE-TREX query set, we create three corresponding *counterfactual training sets* by *removing* one of the following from the FTRACE-TREX abstract set: (a) the most important abstracts for accuracy on $S^\star$, as estimated by TRAK; (b) the abstracts most similar to the queries in $S^\star$ according to BM25; (c) the "ground-truth proponents" for the queries in $S^\star$ as per FTRACE-TREX.

We then measure the average *decrease* in performance on $S^\star$ when a model is finetuned on these counterfactual datasets compared to finetuning on the full training set. Intuition suggests that performance should decrease the most when models are trained on counterfactual training set (c); there is ostensibly *no* direct evidence for *any* of the facts corresponding to the queries in $S^\star$ anywhere in that set.

We find (see Figure 5), however, that it is only the TRAK-*based counterfactual training set* that causes a large change in model behavior. That is, removing abstracts identified with TRAK leads to a 34% decrease in accuracy, significantly more than the decreases induced by removing abstracts according to BM25 (10%) or even removing *ground-truth proponents* (12%).

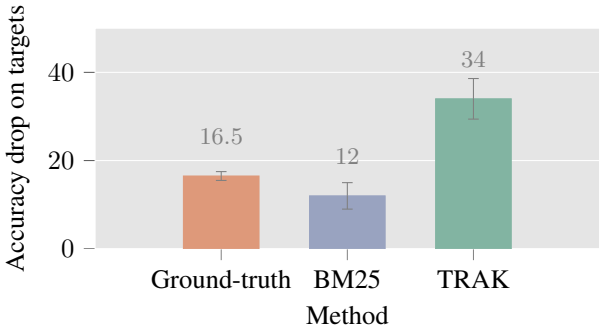

Figure 5. Identifying counterfactually important examples for learning facts on FTRACE-TREX. We compare how three different interventions—removing abstracts with the highest TRAK scores, removing the most similar abstracts according to BM25, and removing the ground-truth proponents as indicated by FTRACE-TREX—affect the resulting model's accuracy on the queries that are answered correctly prior to intervention. The y-axis shows the *decrease* in accuracy (relative to the original model) after each intervention; results are averaged over 50 queries and eight independent models. Error bars represent 95% confidence intervals.

Discussion. Our results demonstrate that while TRAK may not be effective at identifying abstracts that directly express the same fact as a given query, it *can* successfully identify the abstracts that are most responsible for the finetuned model *learning* a given fact. In particular, our analysis suggests that TRAK's subpar performance on the attribution benchmark is an artifact of the FTRACE-TREX benchmark rather than a flaw of TRAK itself. We discuss several potential explanations for this phenomenon in Appendix H.5.

More broadly, our results highlight a difference between *fact tracing* and *behavior tracing*. In other words, finding a data source that supports a given model-generated text is a different task than identifying the actual data sources that *caused* the model to generate this text. While we may be able to address the former problem with model-independent techniques such as information retrieval or web search, the latter requires methods that remain faithful to (and thus, dependent on) the model being studied. Our results here indicate that TRAK can be an effective tool for the latter.

6. Discussion & Conclusion

In our work, we formalize the problem of data attribution and introduce a new method, TRAK, that is effective and efficiently scalable. We then demonstrate the usefulness of TRAK in a variety of large-scale settings: image classifiers trained on CIFAR and ImageNet, language models (BERT and mT5), and image-text models (CLIP).

Still, TRAK is not without limitations: in particular, it requires the model to be differentiable, and its effectiveness also depends on the suitability of the linear approximation. That said, the success of the applying the NTK on language modeling tasks (Malladi et al., 2022) as well as our own experiments both suggest that this approximation is likely to continue to work for larger models. TRAK presents a unique opportunity to reap the benefits of data attribution in previously untenable domains, such as large generative models. We discuss possible future work in Appendix I.

Acknowledgements

We thank Ekin Akyurek for help installing and using the FTRACE-TREX benchmark. Work supported in part by the NSF grants CNS-1815221 and DMS-2134108, and Open Philanthropy. This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001120C0015. Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

References

Achille, A., Golatkar, A., Ravichandran, A., Polito, M., and Soatto, S. Lqf: Linear quadratic fine-tuning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Agarwal, N., Bullins, B., and Hazan, E. Second-order stochastic optimization for machine learning in linear time. In *The Journal of Machine Learning Research*, 2017.

Akyurek, E., Bolukbasi, T., Liu, F., Xiong, B., Tenney, I., Andreas, J., and Guu, K. Towards tracing factual knowledge in language models back to the training data. In *Findings of EMNLP*, 2022.

Alaa, A. and Van Der Schaar, M. Discriminative jackknife: Quantifying uncertainty in deep learning via higher-order influence functions. In *International Conference on Machine Learning*, 2020.

Arnoldi, W. E. The principle of minimized iterations in the solution of the matrix eigenvalue problem. In *Quarterly of applied mathematics*, 1951.

Arora, S., Du, S. S., Hu, W., Li, Z., and Wang, R. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *International Conference on Machine Learning (ICML)*, 2019.

Atanasov, A., Bordelon, B., and Pehlevan, C. Neural networks as kernel learners: The silent alignment effect. In *ICLR*, 2022.

Atanasov, A., Bordelon, B., Sainathan, S., and Pehlevan, C. The onset of variance-limited behavior for networks in the lazy and rich regimes. In *ICLR*, 2023.

Bachmann, G., Hofmann, T., and Lucchi, A. Generalization through the lens of leave-one-out error. In *arXiv preprint arXiv:2203.03443*, 2022.

Bae, J., Ng, N., Lo, A., Ghassemi, M., and Grosse, R. If influence functions are the answer, then what is the question? In *ArXiv preprint arXiv:2209.05364*, 2022.

Bai, Y. and Lee, J. D. Beyond linearization: On quadratic and higher-order approximation of wide neural networks. In *ICLR*, 2020.

Basu, S., You, X., and Feizi, S. Second-order group influence functions for black-box predictions. In *International Conference on Machine Learning (ICML)*, 2019.

Basu, S., Pope, P., and Feizi, S. Influence functions in deep learning are fragile. In *International Conference on Learning Representations (ICLR)*, 2021.

Blum, A. Random projection, margins, kernels, and feature-selection. In *Lecture notes in computer science*. Springer, 2006.

Bohnet, B., Tran, V. Q., Verga, P., Aharoni, R., Andor, D., Soares, L. B., Eisenstein, J., Ganchev, K., Herzig, J., Hui, K., et al. Attributed question answering: Evaluation and modeling for attributed large language models. In *Arxiv preprint arXiv:2212.08037*, 2022.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., and Zhang, C. Quantifying memorization across neural language models. In *arXiv preprint arXiv:2202.07646*, 2022.

Chang, T.-Y. and Jia, R. Careful data curation stabilizes in-context learning. In *Arxiv preprint arXiv:2212.10378*, 2022.

Cherti, M., Beaumont, R., Wightman, R., Wortsman, M., Ilharco, G., Gordon, C., Schuhmann, C., Schmidt, L., and Jitsev, J. Reproducible scaling laws for contrastive language-image learning. In *arXiv preprint arXiv:2212.07143*, 2022.

D'Amour, A., Heller, K. A., Moldovan, D., Adlam, B., Alipanahi, B., Beutel, A., Chen, C., Deaton, J., Eisenstein, J., Hoffman, M. D., Hormozdiari, F., Houlsby, N., Hou, S., Jerfel, G., Karthikesalingam, A., Lucic, M., Ma, Y., McLean, C. Y., Mincu, D., Mitani, A., Montanari, A., Nado, Z., Natarajan, V., Nielson, C., Osborne, T. F., Raman, R., Ramasamy, K., Sayres, R., Schrouff, J., Seneviratne, M., Sequeira, S., Suresh, H., Veitch, V., Vladymyrov, M., Wang, X., Webster, K., Yadlowsky, S., Yun, T., Zhai, X., and Sculley, D. Underspecification presents challenges for credibility in modern machine learning. In *Arxiv preprint arXiv:2011.03395*, 2020.

Deshpande, A., Achille, A., Ravichandran, A., Li, H., Zancato, L., Fowlkes, C., Bhotika, R., Soatto, S., and Perona, P. A linearized framework and a new benchmark for model selection for fine-tuning. In *arXiv preprint arXiv:2102.00084*, 2021.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

Donoho, D. L. De-noising by soft-thresholding. In *IEEE Transactions on Information Theory*, 1995.

Elsahar, H., Vougiouklis, P., Remaci, A., Gravier, C., Hare, J., Laforest, F., and Simperl, E. T-rex: A large scale alignment of natural language with knowledge base triples. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

Fang, A., Ilharco, G., Wortsman, M., Wan, Y., Shankar, V., Dave, A., and Schmidt, L. Data determines distributional robustness in contrastive language image pre-training (clip). In *ICML*, 2022.

Feldman, V. and Zhang, C. What neural networks memorize and why: Discovering the long tail via influence estimation. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, pp. 2881–2891, 2020.

Gao, R., Cai, T., Li, H., Hsieh, C.-J., Wang, L., and Lee, J. D. Convergence of adversarial training in overparametrized neural networks. In *Advances in Neural Information Processing Systems*, 2019.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations (ICLR)*, 2019.

Ghorbani, A. and Zou, J. Data shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning (ICML)*, 2019.

Gu, T., Dolan-Gavitt, B., and Garg, S. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

Hammoudeh, Z. and Lowd, D. Identifying a training-set attack's target using renormalized influence estimation. In *arXiv preprint arXiv:2201.10055*, 2022a.

Hammoudeh, Z. and Lowd, D. Training data influence analysis and estimation: A survey. In *arXiv preprint arXiv:2212.04612*, 2022b.

Hampel, F. R., Ronchetti, E. M., Rousseeuw, P. J., and Stahel, W. A. *Robust statistics: the approach based on influence functions*, volume 196. John Wiley & Sons, 2011.

Hanawa, K., Yokoi, S., Hara, S., and Inui, K. Evaluation of similarity-based explanations. In *International Conference on Learning Representations (ICLR)*, 2021.

Hellmann, S., Lehmann, J., Auer, S., and Brümmer, M. Integrating nlp using linked data. In *The Semantic Web–ISWC 2013: 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part II 12*, pp. 98–113. Springer, 2013.

Ho, J., Jain, A., and Abbeel, P. Denoising diffusion probabilistic models. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., Casas, D. d. L., Hendricks, L. A., Welbl, J., Clark, A., et al. Training compute-optimal large language models. In *arXiv preprint arXiv:2203.15556*, 2022.

Holzmüller, D., Zaverkin, V., Kästner, J., and Steinwart, I. A framework and benchmark for deep batch active learning for regression. *arXiv preprint arXiv:2203.09410*, 2022.

Huang, J. and Yau, H.-T. Dynamics of deep neural networks and neural tangent hierarchy. In *Proceedings of the 37th International Conference on Machine Learning*, 2020.

Huh, M., Agrawal, P., and Efros, A. A. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., and Madry, A. Adversarial examples are not bugs, they are features. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Ilyas, A., Park, S. M., Engstrom, L., Leclerc, G., and Madry, A. Datamodels: Predicting predictions from training data. In *International Conference on Machine Learning (ICML)*, 2022.

Jacot, A., Gabriel, F., and Hongler, C. Neural tangent kernel: Convergence and generalization in neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Jia, R., Dao, D., Wang, B., Hubis, F. A., Hynes, N., Gürel, N. M., Li, B., Zhang, C., Song, D., and Spanos, C. J. Towards efficient data valuation based on the shapley value. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, 2019.

Jia, R., Wu, F., Sun, X., Xu, J., Dao, D., Kailkhura, B., Zhang, C., Li, B., and Song, D. Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

Johnson, W. B. and Lindenstrauss, J. Extensions of lipschitz mappings into a hilbert space. In *Contemporary mathematics*, 1984.

Khanna, R., Kim, B., Ghosh, J., and Koyejo, S. Interpreting black box predictions using fisher kernels. In *The 22nd International Conference on Artificial Intelligence and Statistics*, 2019.

Koh, P. W. and Liang, P. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*, 2017.

Koh, P. W., Ang, K.-S., Teo, H. H., and Liang, P. On the accuracy of influence functions for measuring group effects. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Kong, S., Shen, Y., and Huang, L. Resolving training biases via influence-based data relabeling. In *International Conference on Learning Representations (ICLR)*, 2022.

Krizhevsky, A. Learning multiple layers of features from tiny images. In *Technical report*, 2009.

Leclerc, G. and Madry, A. The two regimes of deep network training. In *arXiv preprint arXiv:2002.10376*, 2020.

Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. Deduplicating training data makes language models better. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.

Lewkowycz, A., Bahri, Y., Dyer, E., Sohl-Dickstein, J., and Gur-Ari, G. The large learning rate phase of deep learning: the catapult mechanism. In *arXiv preprint arXiv:2003.02218*, 2020.

Lin, J., Zhang, A., Lecuyer, M., Li, J., Panda, A., and Sen, S. Measuring the effect of training data on deep learning predictions via randomized experiments. *arXiv preprint arXiv:2206.10013*, 2022.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. Microsoft coco: Common objects in context. In *European conference on computer vision (ECCV)*, 2014.

Liu, Z., Ding, H., Zhong, H., Li, W., Dai, J., and He, C. Influence selection for active learning. In *ICCV*, 2021.

Long, P. M. Properties of the after kernel. In *arXiv preprint arXiv:2105.10585*, 2021.

Lundberg, S. and Lee, S.-I. A unified approach to interpreting model predictions. In *Neural Information Processing Systems (NeurIPS)*, 2017.

Ma, J., Guo, L., and Fattahi, S. Behind the scenes of gradient descent: A trajectory analysis via basis function decomposition. In *arXiv preprint arXiv:2210.00346*, 2022.

Maddox, W., Tang, S., Moreno, P., Wilson, A. G., and Damianou, A. Fast adaptation with linearized neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2021.

Maillard, O. and Munos, R. Compressed least-squares regression. In *Advances in Neural Information Processing Systems*, 2009.

Malladi, S., Wettig, A., Yu, D., Chen, D., and Arora, S. A kernel-based view of language model fine-tuning. In *arXiv preprint arXiv:2210.05643*, 2022.

Mu, F., Liang, Y., and Li, Y. Gradients as features for deep representation learning. In *ICLR*, 2020.

Nguyen, T., Novak, R., Xiao, L., and Lee, J. Dataset distillation with infinitely wide convolutional networks. In *Advances in Neural Information Processing Systems*, 2021a.

Nguyen, T., Raghu, M., and Kornblith, S. Do wide and deep networks learn the same things? uncovering how neural network representations vary with width and depth. In *International Conference on Learning Representations (ICLR)*, 2021b.

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., and Miller, A. Language models as knowledge bases? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

Pregibon, D. Logistic regression diagnostics. In *The Annals of Statistics*, 1981.

Pruthi, G., Liu, F., Sundararajan, M., and Kale, S. Estimating training data influence by tracing gradient descent. In *Neural Information Processing Systems (NeurIPS)*, 2020.

Rad, K. R. and Maleki, A. A scalable estimate of the extra-sample prediction error via approximate leave-one-out. In *ArXiv preprint arXiv:1801.10243*, 2018.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. Learning transferable visual models from natural language supervision. In *arXiv preprint arXiv:2103.00020*, 2021.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 2020.

Rahimi, A. and Recht, B. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, 2007.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Santurkar, S., Tsipras, D., and Madry, A. Breeds: Benchmarks for subpopulation shift. In *International Conference on Learning Representations (ICLR)*, 2021.

Santurkar, S., Dubois, Y., Taori, R., Liang, P., and Hashimoto, T. Is a caption worth a thousand images? a controlled study for representation learning. In *arXiv preprint arXiv:2207.07635*, 2022.

Saunshi, N., Gupta, A., Braverman, M., and Arora, S. Understanding influence functions and datamodels via harmonic analysis. In *ICLR*, 2023.

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

Shah, H., Park, S. M., Ilyas, A., and Madry, A. Modeldiff: A framework for comparing learning algorithms. In *arXiv preprint arXiv:2211.12491*, 2022.

Spearman, C. The proof and measurement of association between two things. In *The American Journal of Psychology*, 1904.

Teso, S., Bontempelli, A., Giunchiglia, F., and Passerini, A. Interactive label cleaning with example-based explanations. In *Advances in Neural Information Processing Systems*, 2021.

Thanei, G.-A., Heinze, C., and Meinshausen, N. Random projections for large-scale regression. In *Big and Complex Data Analysis: Methodologies and Applications*, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Wei, A., Hu, W., and Steinhardt, J. More than a toy: Random matrix models predict how real-world neural representations generalize. In *ICML*, 2022.

Wei, C., Lee, J. D., Liu, Q., and Ma, T. Regularization matters: Generalization and optimization of neural nets vs their induced kernel. In *Advances in Neural Information Processing Systems*, 2019.

Wojnowicz, M., Cruz, B., Zhao, X., Wallace, B., Wolff, M., Luan, J., and Crable, C. Influence sketching: Finding influential samples in large-scale regressions. In *2016 IEEE International Conference on Big Data (Big Data)*, 2016.

Yang, G. and Littwin, E. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In *Proceedings of the 38th International Conference on Machine Learning*, 2021.

Yeh, C.-K., Kim, J. S., Yen, I. E. H., and Ravikumar, P. Representer point selection for explaining deep neural networks. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Yu, Y., Wei, A., Karimireddy, S. P., Ma, Y., and Jordan, M. I. Tct: Convexifying federated learning using bootstrapped neural tangent kernels. In *NeurIPS*, 2022.

Zhang, R. and Zhang, S. Rethinking influence functions of neural networks in the over-parameterized regime. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2022.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. The unreasonable effectiveness of deep features as a perceptual metric. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.

Zinkevich, M. A., Davies, A., and Schuurmans, D. Holographic feature representations of deep networks. In *UAI*, 2017.

Appendices

A. Accelerating Datamodel Applications with TRAK

Our evaluation in the main paper shows taht data attribution scores computed with TRAK can *predict* how a given model's output changes as a function of the composition of the corresponding model's training set. While the capability to make such predictions is useful in its own right, prior work has shown that this primitive also enables many downstream applications (Koh & Liang, 2017; Jia et al., 2019; Alaa & Van Der Schaar, 2020). For example, prior works leverage datamodel scores to identify brittle predictions (Ilyas et al., 2022) and to compare different learning algorithms (Shah et al., 2022). We now show that using TRAK in place of datamodel scores can significantly speed up these downstream applications too.

A.1. Estimating prediction brittleness

Ilyas et al. (2022) use datamodel scores to provide *lower bounds* on the *brittleness* of a given example—that is, given an example of interest z, they identify a subset of the training set whose removal from the training data causes the resulting re-trained model to misclassify z. The brittleness estimation algorithm that Ilyas et al. (2022) leverage hinges on the fact that the datamodel attribution function $\tau_{\text{DM}}(z)$ can accurately predict model outputs, i.e., achieve high LDS. Motivated by TRAK's good performance on the linear datamodeling task (see, e.g., Figure 8), we examine estimating the brittleness of CIFAR-10 examples using TRAK scores in place of datamodel ones (but otherwise following the procedure of Ilyas et al. (2022)). Our results (see Figure 6) indicate that TRAK scores computed from an ensemble of just 100 models are about as effective at estimating brittleness as datamodel scores computed from 50,000 models. Thus, TRAK scores can be a viable (and orders of magnitude faster) alternative to datamodels for estimating prediction brittleness.

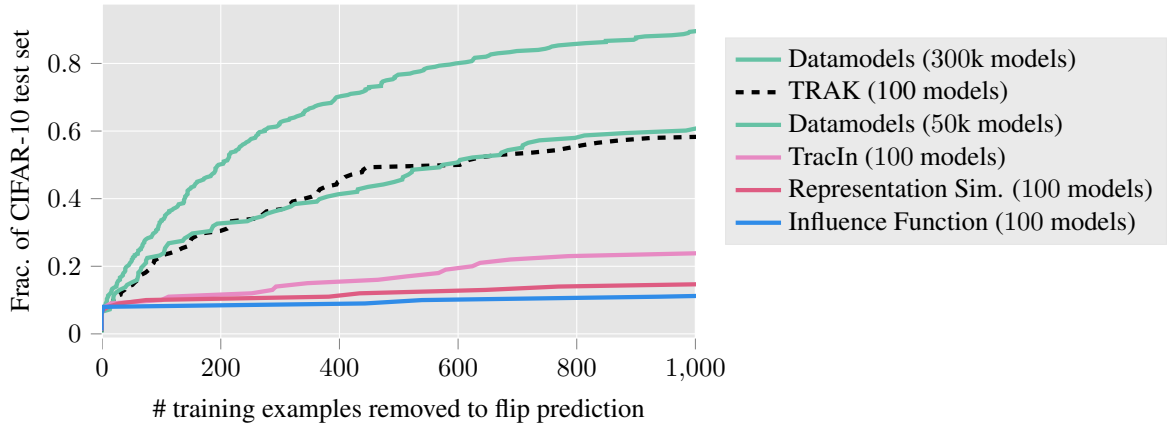

Figure 6. Using TRAK *scores to identify brittle model predictions.* Following the methodology of Ilyas et al. (2022), we apply different data attribution methods to estimate the brittleness of model predictions on examples from the CIFAR-10 validation set. The number of models used by each attribution method is specified in parentheses, e.g., TRAK (100) indicates that TRAK scores were computed using an ensemble of 100 trained models.

A.2. Learning algorithm comparisons

A useful way to leverage datamodels is to view them as *data representations*. More specifically, following Ilyas et al. (2022), for an example of interest z, one can view the datamodel attribution $\tau_{\text{DM}}(z)$ as an embedding of z into $\mathbb{R}^n$, where n is the size of the training dataset. Analyzing examples in such induced *datamodel representation spaces* turns out to enable uncovering dataset biases and model-specific subpopulations (Ilyas et al., 2022). Furthermore, this representation space is not specific to a particular model instance or architecture—it is *globally aligned* in the sense that for the same example z, the attribution score $\tau_{\text{DM}}(z)_i$ of a given train example i has a consistent interpretation across *different* learning pipelines. Shah et al. (2022) leverage the properties of the datamodel representation space to perform model-agnostic *learning algorithm comparison* (called MODELDIFF): given two learning algorithms, they show how to use datamodels to identify *distinguishing features*, i.e., features that are used by one learning algorithm but not the other.

Once again, motivated by TRAK's good performance on the LDS metric, we investigate whether TRAK scores can substitute for datamodel scores in this context. To this end, we revisit one of the case studies from Shah et al. (2022)—the one that compares image classifiers trained with and without data augmentation, and identifies features that distinguish these two

classes of models. When applied to this case study, MODELDIFF computed with TRAK scores recovers similar distinguishing features to the ones originally found by Shah et al. (2022) (using datamodel scores)—see Figure 7 for more details. Also, employing TRAK scores in place of datamodel scores reduces the total computational cost by a factor of 100, showing, once again, that TRAK can dramatically accelerate downstream tasks that rely on accurate attribution scores.

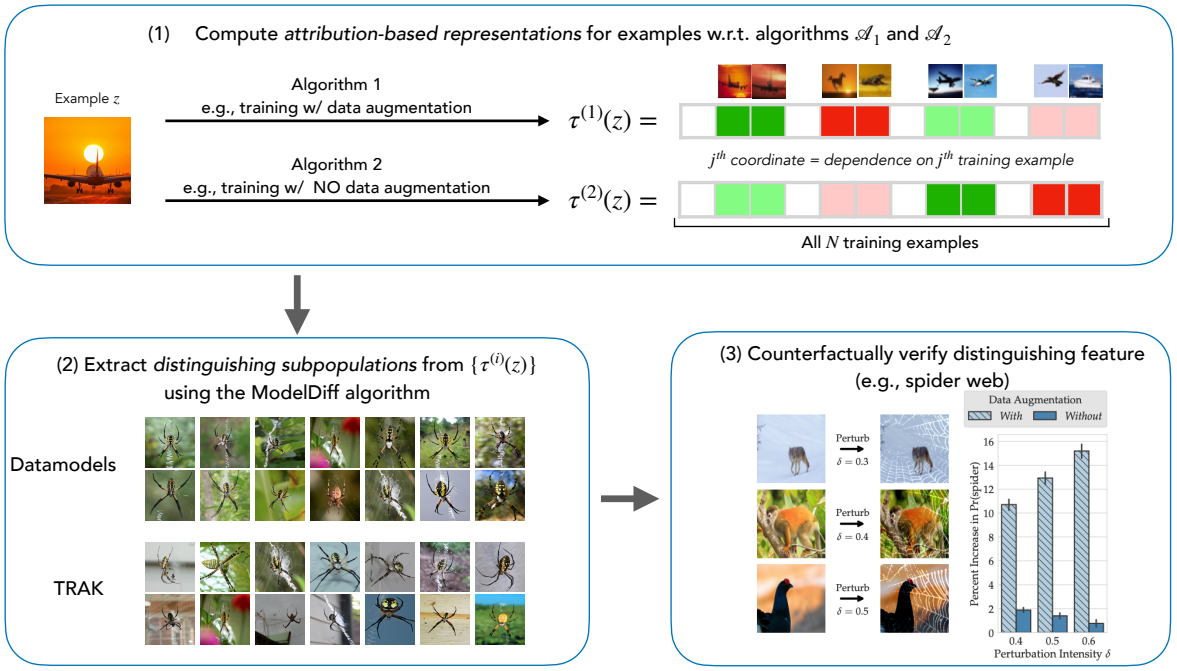

Figure 7. Accelerating learning algorithm comparisons with TRAK. The MODELDIFF framework from (Shah et al., 2022) uses datamodel representations to surface features that distinguish two learning algorithms. In the case study here, we compare models trained on the LIVING17 dataset *with* and *without* data augmentation. Applying MODELDIFF involves three stages: (1) computing datamodel representations; (2) applying the MODELDIFF algorithm to extract *distinguishing subpopulations* of inputs on which two model classes behave differently; (3) counterfactually testing the inferred feature associated with the subpopulation. Shah et al. (2022) find that models trained with data augmentation latch onto the presence of spider webs as a spurious correlation to predict the class spider. Here, we recover their result by using TRAK scores instead of datamodel scores in step (1); doing so reduces the computational cost of MODELDIFF by 100x.

B. Related work

In this section, we highlight and discuss how TRAK connects to prior works on training data attribution, the neural tangent kernel, and kernel approximation.

Training data attribution. There is a sizable body of work on data attribution methods. Here we discuss approaches most similar to ours, but we refer the reader back to Section 2 for an overview of prior work on data attribution methods and to (Hammoudeh & Lowd, 2022b) for an even more extensive survey.

In the setting of generalized linear models, Wojnowicz et al. (2016) speed up classical influence estimation (Definition 3.1) by leveraging random projections. Also, Khanna et al. (2019) employ a similar estimator based on the Fisher matrix for data attribution and subset selection. Their experiments are limited though to small neural networks and linear models. Most similarly to our approach, Achille et al. (2021) leverage the linearized model for approximating influence functions (among other applications). However, their approach introduces several changes to the model of interest (such as modifying activations, loss, and regularization) and focuses on finetuning in smaller-scale settings, whereas TRAK can be applied directly to the original model (and at scale).

Similarly to us, prior works also investigate the tradeoffs between scalability and efficacy of data attribution methods. For instance, Jia et al. (2021) study these tradeoffs by proposing new metrics and comparing according to them leave-one-out methods (e.g., influence functions) and Shapley values. They put forth, in particular, a new estimator for Shapley values that is based on approximating the original model with a k-nearest neighbors model over the pre-trained embeddings—this can be viewed as an alternative to working with the linearized model.

As discussed in Section 2, a major line of work uses *Hessian-based influence functions* for data attribution (Koh & Liang, 2017; Koh et al., 2019; Basu et al., 2021). In particular, the influence function effectively computes—up to an error that can be bounded—the one-step Newton approximation with respect to the full model parameters (Koh et al., 2019). Recall that TRAK also leverages the one-step Newton approximation in order to estimate leave-one-out influences for logistic regression (see Section 3). However, in contrast to the influence function approach, the Hessian matrix we leverage (the matrix $X^\top RX$ in Definition 3.1) is positive semi-definite as it is computed with respect to the *linearized model* rather than the original model. As a result, computing TRAK does not require the use of additional regularization (beyond the one implicitly induced by our use of random projections), which is practically necessary in the influence function approach. Prior works also leverage a similar Hessian matrix based on the generalized Gauss-Newton matrix (Bae et al., 2022) or the equivalent Fisher information matrix (Teso et al., 2021), which are guaranteed to be positive semi-definite.

Neural tangent kernel. The neural tangent kernel (NTK) (Jacot et al., 2018) and its generalizations (Yang & Littwin, 2021) are widely studied as a tool for theoretically analyzing generalization (Arora et al., 2019), optimization (Wei et al., 2019), and robustness (Gao et al., 2019) of (overparameterized) neural networks. While these works focus on neural networks in the their large or infinite-width limit, a line of recent works (Mu et al., 2020; Achille et al., 2021; Long, 2021; Atanasov et al., 2022; Wei et al., 2022; Malladi et al., 2022; Atanasov et al., 2023; Ma et al., 2022) studies instead the finite-width *empirical NTK* (eNTK). Our TRAK estimator is partly motivated by the observation from this line of work that kernel regression with the eNTK provides a good approximation to the original model.

While we leverage the eNTK approximation for data attribution, prior works leveraged the NTK and eNTK for various other applications, such as studying generalization (Bachmann et al., 2022), sample selection for active learning (Holzmüller et al., 2022), model selection (Deshpande et al., 2021), federated learning (Yu et al., 2022), and fast domain adaptation (Maddox et al., 2021). Our reduction to the linear case (Step 1 in Section 3.2) is analogous to the approach of Bachmann et al. (2022) that leverages formulas for the leave-one-out error of kernel methods coupled with the NTK approximation to estimate the generalization error. Another related work is that of Zhang & Zhang (2022), who theoretically characterize the accuracy of the Hessian-based influence function in the NTK regime (i.e., large-width limit).

Finally, although the work on NTK popularized the idea of leveraging gradients as features, similar ideas can be traced back to works on the Fisher kernel and related ideas (Zinkevich et al., 2017).

Kernel methods and random projections. Our application of random projections to improve computational efficiency of kernel approximation is a widely used idea in kernel methods (Blum, 2006; Rahimi & Recht, 2007). Aside from computational advantages, this technique can also provide insight into empirical phenomena. For example, Malladi et al. (2022) use the kernel view along with random projections as a lens to explain the efficacy of subspace-based finetuning methods.

C. Experimental Setup

C.1. Datasets and models

CIFAR. We construct the CIFAR-2 dataset as the subset of CIFAR-10 (Krizhevsky, 2009) consisting of only the "cat" and "dog" classes. We initially used CIFAR-2 as the main test bed when designing TRAK, as it is a binary classification task and also smaller in size. On both CIFAR-2 and CIFAR-10, we train a ResNet-9 architecture.[9] For CIFAR-2, we use (max) learning rate 0.4, momentum 0.9, weight decay 5e-4, and train for 100 epochs using a cyclic learning rate schedule with a single peak at epoch 5. For CIFAR-10, we replace the learning rate with 0.5 and train for 24 epochs.

Our code release includes a notebook[10] that can reproduce the CIFAR-2 results end-to-end. **ImageNet.** We use the full 1000-class ImageNet dataset and train a modified ResNet-18 architecture. Models are trained from scratch for 15 epochs, cyclic learning rate with peak at epoch 2 and initial learning rate 5.2, momentum 0.8, weight decay 4e-5, and label smoothing 0.05.

QNLI. We finetune a pre-trained BERT model (`bert-base-cased`[11]) on the QNLI (Question-answering Natural Language Inference) task from the GLUE benchmark. We use the default training script[12] from HuggingFace with a few modifications: we use SGD (20 epochs, learning rate starting at 1e-3) instead of AdamW, and we remove the last `tanh` non-linearity before the classification layer. Removing the last non-linearity prevents the model outputs in saturating, resulting in higher LDS. (That said, we find that TRAK scores can be still computed on the models with non-linearity; this was only for improving evaluation.) We restrict the training set to 50,000 examples, approximately half of the full training set.

CLIP on MS COCO. We use an open-source implementation[13] of CLIP. The model uses a ResNet-50 for the image encoder and a Transformer for the text encoder (for captions). We train for 100 epochs using the Adam optimizer with batch size 600, a cosine learning rate schedule with starting learning rate 0.001, weight decay 0.1, and momentum 0.9. All images are resized to a resolution of 224×224. We use random resize crop, random horizontal flip, and Gaussian blur as data augmentations.

In the counterfactual evaluation, we consider a normalized notion of cosine similarity, $\bar{r} = r/(r_{95} - r_5)$, where r is the raw correlation between image and caption embeddings and r_α is the α-percentile of image-caption similarities across the entire dataset. Results remain similar with other choices of metric.

Fact tracing mT5 on FTRACE-TREX. We follow the setup exactly as in Akyurek et al. (2022) as we describe in Section 5.2, other than using a smaller architecture (`mt5-small`). See Appendix H for more details.

MODELDIFF on LIVING17. The LIVING17 dataset (Santurkar et al., 2021) is an image classification dataset derived from the ImageNet dataset and consists of 17 classes, each comprised of four original ImageNet classes.

We train the standard ResNet-18 architecture on the above dataset, either using standard data augmentation (random resized cropping and random horizontal flips) or with no data augmentation (only center cropping, same as used on when evaluating). The goal of the case study from Shah et al. (2022) is to distinguish the above two learning algorithms in terms of the feature priors of the resulting trained models. To run MODELDIFF, follow the setup in Shah et al. (2022) exactly; we refer to the work for more details of the case study and implementation details.

C.2. TRAK hyperparameters

TRAK only has two hyperparmeters: the projection dimension k and the number of models M. The following hyperparameters were used unless specified otherwise:

Soft-thresholding. An optional hyperparameter is needed if we use soft-thresholding (Step 5). Among the four tasks we evaluate the LDS on, we find that soft-thresholding is only helpful for the non-binary classification tasks (i.e., CIFAR-10 and

[9] https://github.com/wbaek/torchskeleton/blob/master/bin/dawnbench/cifar10.py
[10] https://github.com/MadryLab/trak/blob/main/examples/cifar2_correlation.ipynb
[11] https://huggingface.co/bert-base-cased
[12] https://github.com/huggingface/transformers/blob/main/examples/pytorch/text-classification/run_glue.py
[13] https://github.com/mlfoundations/open_clip

Dataset	Model	Number of models (M)	Projection dimension (k)
CIFAR-2	ResNet-9	-	4,000
CIFAR-10	ResNet-9	-	20,000
QNLI	BERT-BASE	-	4,000
ImageNet	ResNet-18	-	15,000
MS COCO	ResNet-50 (CLIP)	100	20,000
FTRACE-TREX	`mt5-small`	10	4,000
LIVING-17	ResNet-18	100	1,000

Table 1. TRAK hyperparameters used for different experiments. Blank indicates that different numbers were used depending on the experiment.

ImageNet, but not CIFAR-2 and QNLI); intuitively, this may be due to the fact that the underlying model output function depends on fewer examples (i.e., the attribution vector is sparser) when there are more classes.

For both CIFAR-10 and ImageNet, we use a single sparsity threshold—i.e., for each test example, we choose the soft-thresholding parameter λ s.t. the resulting TRAK score vector has exactly k non-zero entries, and use the same k for all test examples. To choose k, for CIFAR-10 we cross-validate using the same M models that we used to compute TRAK scores, when $M \geq 20$; in other words, we avoid "cheating" by using additional models for cross-validation. For ImageNet, we simply choose $k = 1000$ since there are on average 1,300 training examples per class.

C.3. Baselines

We provide details on baselines used in our evaluation in Section 4. Though most of the existing approximation-based methods only use a single model checkpoint in their original formulation, we average the methods over multiple independent checkpoints to help increase its performance.

Influence functions. The standard Hessian-based influence functions yield the attribution scores

$$\tau(z_j)_i = \nabla L(z_j; \theta^\star)\ H_{\theta^\star}^{-1}\ \nabla L(z_i; \theta^\star),$$

where $H_{\theta^\star}$ is the empirical Hessian w.r.t. the training set. We use an existing PyTorch implementation[14] that uses the stochastic approximation of inverse-Hessian-vector products using the LISSA (Agarwal et al., 2017) algorithm as done in Koh & Liang (2017). As in the original work, we compute the gradients only with respect to the last linear layer; using additional layers caused the inversion algorithm to either diverge or to run out of memory. For hyperparameters, we use similar values as done in prior work; we use $r = 1$, $d = 5000$, and damping factor of 0.01. We find that additional repeats (r, the number of independent trials to average each iHvp estimate) does not help, while increasing the depth (d, the number of iterations used by LISSA) helps significantly.

Influence functions based on the Arnoldi iteration. This variant of influence functions from Schioppa et al. (2022) is based on approximating the top eigenspace of the Hessian using the Arnoldi iteration (Arnoldi, 1951). We use the original implementation in JAX.[15] We normalize the gradients as recommended in the original paper. While much faster than the original formulation in Koh & Liang (2017), we find that the attribution scores not very predictive (according to the LDS).

TracIn. We use the TracInCP estimator from (Pruthi et al., 2020), defined as

$$\tau(z_j)_i = \sum_{t=1}^{T} \eta_t \cdot \nabla L(z_j; \theta_t) \cdot \nabla L(z_i; \theta_t),$$

where θ_t is the checkpoint from the epoch t and η_t is the corresponding learning rate η_t. We also average over trajectories of multiple independently trained models, which increases its performance. We approximate the dot products using random projections of dimensions 500-1000 as we do for TRAK, as the estimator is intractable otherwise. We found that increasing the number of samples (epochs) from the training trajectory does not lead to much improvement.

[14] https://github.com/alstonlo/torch-influence
[15] https://github.com/google-research/jax-influence

Gradient Aggregated Similarity (GAS). This is a "renormalized" version of the TracInCP (Hammoudeh & Lowd, 2022b) based on using the cosine similarity instead of raw dot products. In general, its performance is indistinguishable from that of TracIn.

Representation similarity. We use the *signed* ℓ_2 dot product in representation space (feature embeddings of the penultimate layer), where the sign indicates whether the labels match. We also experimented with cosine similarity but the resulting performance was similar.

Empirical influences. We use the subsampling-based approximation to leave-one-out influences as used by (Feldman & Zhang, 2020), which is a difference-in-means estimator given by

$$\tau(z_j)_i = \mathbb{E}_{S \ni z_i} f(z_j; \theta) - \mathbb{E}_{S \not\ni z_i} f(z_j; \theta)$$

where the first (second) expectation is over training subsets that include (exclude) example z_i.

Datamodels. We use the ℓ_1-regularized regression-based estimators from Ilyas et al. (2022), using up to 60,000 models for CIFAR-2 and 300,000 models for CIFAR-10 (trained on different random 50% subsets of the full training set).

C.4. Linear Datamodeling Score

Let τ be a given data attribution method (as framed in Definition 2.1), and let $g_\tau(z, S'; S)$ be its corresponding attribution-derived prediction function (see Definition 2.2). Then, to evaluate τ:

1. We sample 100 different random subsets $\{S_j \subset S : j \in [100]\}$ of the training set S, and train five models on each one of these subsets. Each subset S_j is sampled to be 50% of the size of S, but we also consider other subsampling ratios in Appendix F.

2. For each example of interest z (i.e., for each example in the test set of the dataset we are studying), we approximate the expectation of the model output $\mathbb{E}[f(z; \theta_i^\star(S_j))]$ for each training subset S_j (where the expectation is taken over the learning algorithm's randomness) by averaging across the corresponding five models $\{\theta_i^\star(S_j)\}_{i=1}^5$.

3. We then compute the linear datamodeling score for each example of interest z as the Spearman rank correlation between the averaged model outputs computed in the previous step and the attribution-derived predictions $g_\tau(z, S_j; S)$ of model outputs. That is, we compute:

$$\text{Spearman-}\rho\Bigg(\underbrace{\left\{\frac{1}{5}\sum_{i=1}^{5} f(z; \theta_i^\star(S_j)) : j \in [100]\right\}}_{\text{averaged model outputs}}, \underbrace{\{g_\tau(z, S_j; S) : j \in [100]\}}_{\substack{\text{attributed-derived predictions} \\ \text{of model outputs}}}\Bigg)$$

4. Finally, we average the LDS (Definition 2.3) across 2,000 examples of interest, sampled at random from the validation set, and report this score along with the 95% bootstrap confidence intervals corresponding to the random re-sampling from the subsets S_j.

C.5. Hardware and wall-time measurements

For all of our experiments, we use NVIDIA A100 GPUs each with 40GB of memory and 12 CPU cores. We evaluate the computational cost of attribution methods using two metrics, *total wall-time* and the *total number of trained models used*; see Section 4 for motivation behind these metrics. For most attribution methods, one or more of the following components dominate their total runtime:

- `TRAIN_TIME`: the time to train one model (from scratch)

- `GRAD_TIME`: the time to compute gradients of one model (including computing random projections) for the entire dataset under consideration (both train and test sets). This time may vary depending on size of the projection dimension, but our fast implementation (Appendix D) can handle dimensions of up to 80,000 without much increase in runtime.

The total compute time for each method was approximated as follows, where M is the number of models used:

- **TRAK:** $M \times (\texttt{TRAIN\_TIME} + \texttt{GRAD\_TIME})$, as we have to compute gradients for each of the trained models.
- **Datamodel (Ilyas et al., 2022) and Empirical Influence (Feldman & Zhang, 2020):** $M \times \texttt{TRAIN\_TIME}$. The additional cost of estimating datamodels or influences from the trained models (which simply involves solving a linear system) is negligible compared to the cost of training.
- **LiSSA based influence functions (Koh & Liang, 2017):** These approaches are costly because they use thousands of Hessian-vector product iterations to approximate a single inverse-Hessian-vector product (which is needed for each target example). Hence, we computed these attribution scores for a much smaller sample of validation set (50 to 100). We measured the empirical runtime on this small sample and extrapolated to the size of the entire (test) dataset.
- **Influence function based on the Arnoldi iteration (Schioppa et al., 2022):** We ran the authors' original code[16] on CIFAR models of the same architecture (after translating them to `JAX`) and measured the runtime.
- **TracIn (Pruthi et al., 2020) and GAS (Hammoudeh & Lowd, 2022a):** $M \times (\texttt{TRAIN\_TIME} + \texttt{GRAD\_TIME} \times T)$, where T is the number of checkpoints used per model.

[16] https://github.com/google-research/jax-influence

D. TRAK implementation

We release an easy-to-use library, `trak`,[17], which computes TRAK scores using Algorithm 1. Computing TRAK involves the following four steps: (i) training models (or alternatively, acquiring checkpoints), (ii) computing gradients, (iii) projecting gradients with a random projection matrix (Rademacher or Gaussian), and (iv) aggregating into the final estimator (Equation (15)).

Step (i) is handled by the user, while steps (ii)-(iv) are handled automatically by our library. Step (ii) is implemented using the `functorch` library to compute per-example gradients. Step (iii) is either implemented using matrix multiplication on GPU or by a faster custom CUDA kernel, which is described below. Step (iv) just involves a few simple matrix operations.

D.1. Fast random projections on GPU

One of the most costly operation of TRAK is the random projection of the gradients onto a smaller, more manageable vector space. While CPUs are not equipped to handle this task on large models (e.g., LLMs) at sufficient speed, at least on paper, GPUs have more than enough raw compute.

In practice, however, challenges arise. First, storing the projection matrix entirely is highly impractical. For example, a matrix for a model with 300 million weights and an output of 1024 dimensions would require in excess of 1TB of storage. One solution is to generate the projection in blocks (across the output dimension). This solution is possible (and offered in our implementation) but is still radically inefficient. Indeed, even if the generation of the matrix is done by block it still has to be read and written once onto the GPU RAM. This severely limits the performance as memory throughput becomes the bottleneck.

Our approach. Our solution is to generate the coefficients of the projection as needed (in some situations more than once) and never store them. As a result, the bandwidth of the RAM is solely used to retrieve the values of the gradients and write the results at the end. This forces us to use pseudo-randomness but this is actually preferrable since a true random matrix would make experiments impossible to reproduce exactly.

Our implementation is written in C++/CUDA and targets NVIDIA GPUs of compute capability above or equal 7.0 (V100 and newer). It supports (and achieve better performance) batches of multiple inputs, and either normally distributed coefficients or -1, 1 with equal probabilities.

Implementation details. We decompose the input vectors into K blocks, where each block is projected independently to increase parallelism. The final result is obtained by summing each partial projection. To reduce memory usage, we keep K to roughly 100.

We further increase parallelism by spawning a thread for each entry of the output blocks, but this comes at the cost of reading the input multiple times. To mitigate this issue, we use Shared Memory offered by GPUs to share and reduce the frequency of data being pulled from global memory. We also use Shared Memory to reduce the cost of generating random coefficients, which can be reused for all the inputs of a batch.

Finally, we take advantage of Tensor Cores to maximize throughput and efficiency, as they were designed to excel at matrix multiplications. These interventions yield a fast and power-efficient implementation of random projection. On our hardware, we achieved speed-ups in excess of 200x compared to our "block-by-block" strategy.

[17] https://github.com/MadryLab/trak

D.2. Pseudocode

Algorithm 1 TRAK for multi-class classifiers (as implemented)

Input: Learning algorithm $\mathcal{A}$, dataset S of size n, sampling fraction $\alpha \in (0, 1]$, correct-class likelihood function $p(z; \theta)$, projection dimension $k \in \mathbb{N}$
Output: Matrix of attribution scores $\mathbf{T} \in \mathbb{R}^{n \times n}$
$f(z; \theta) := \log(\frac{p(z;\theta)}{1-p(z;\theta)})$ ▷ Margin function f_θ
for $m \in \{1, \ldots, M\}$ **do**
  Sample random $S' \subset S$ of size $\alpha \cdot n$
  $\theta^\star_m \leftarrow \mathcal{A}(S')$ ▷ Train a model on S'
  $\mathbf{P} \sim \mathcal{N}(0, 1)^{p \times k}$ ▷ Sample projection matrix
  $\mathbf{Q}^{(m)} \leftarrow \mathbf{0}_{n \times n}$
  for $i \in \{1, \ldots, n\}$ **do**
    $\phi_i \leftarrow \mathbf{P}^\top \nabla_\theta f(z_i; \theta^\star_m)$ ▷ Compute gradient at $\theta^\star_m$ and project to k dimensions
    $\mathbf{Q}^{(m)}_{ii} \leftarrow 1 - p(z_i; \theta^\star)$ ▷ Compute weighting term
  end for
  $\Phi_m \leftarrow [\phi_1; \cdots; \phi_n]^\top$
end for
$\mathbf{T} \leftarrow \left[\frac{1}{m} \sum_{m=1}^{M} \Phi_m (\Phi_m^\top \Phi_m)^{-1} \Phi_m^\top\right] \left[\frac{1}{m} \sum_{m=1}^{M} \mathbf{Q}^{(m)}\right]$
return SOFT-THRESHOLD($\mathbf{T}$)

E. Theoretical Justification

E.1. The one-step Newton approximation for leave-one-out influence

The key formula we use in TRAK is the estimate for the leave-one-out (LOO) influence in logistic regression (Definition 3.1). Here, we reproduce the derivation of this estimate from Pregibon (1981) then extend it to incorporate example-dependent bias terms.

Convergence condition for logistic regression. Assume that we optimized the logistic regression instance via Newton-Raphson, i.e., the parameters are iteratively updated as

$$\widehat{\theta}_{t+1} \leftarrow \widehat{\theta}_t + H_{\widehat{\theta}_t}^{-1} \nabla_\theta L(\widehat{\theta}_t) \tag{17}$$

where $H_{\widehat{\theta}}$ is the Hessian and $\nabla_\theta L(\widehat{\theta})$ is the gradient associated with the total training loss $L(\widehat{\theta}) = \sum_{z_i \in S} L(z_i; \theta)$. In the case of logistic regression, the above update is given by

$$\widehat{\theta}_{t+1} \leftarrow \widehat{\theta}_t + (X^\top R X)^{-1} X^\top \hat{q} \tag{18}$$

where $\hat{q} = \vec{1} - \hat{p}$ is the vector of the probabilities for the *incorrect* class evaluated at $\widehat{\theta}_t$ and $R = \text{diag}(\hat{p}(1-\hat{p})$ is the corresponding matrix. Upon convergence, the final parameters $\theta^\star$ satisfy the following:

$$(X^\top R X)^{-1} X^\top q^\star = 0 \tag{19}$$

where $q^\star$ is the incorrect-class probability vector corresponding to $\theta^\star$.

The one-step Newton approximation. We estimate the counterfactual parameters $\theta^\star_{-i}$ that would have resulted from training on the same training set excluding example i by simply taking a single Newton step starting from the same global optimum $\theta^\star$:

$$\theta^\star_{-i} = \theta^\star + (X_{-i}^\top R_{-i} X_{-i})^{-1} X_{-i}^\top q^\star_{-i}, \tag{20}$$

where the subscript $-i$ denotes the corresponding matrices and vectors without the i-th training example. Rearranging and using (19),

$$\begin{aligned}
\theta^\star - \theta^\star_{-i} &= -(X_{-i}^\top R_{-i} X_{-i})^{-1} X_{-i}^\top q^\star_{-i} \\
\theta^\star - \theta^\star_{-i} &= (X^\top R X)^{-1} X^\top q^\star - (X_{-i}^\top R_{-i} X_{-i})^{-1} X_{-i}^\top q^\star_{-i}
\end{aligned}$$

Using the Sherman–Morrison formula to simplify above,[18] we have

$$\theta^\star - \theta^\star_{-i} = \frac{(X^\top R X)^{-1} x_i}{1 - x_i^\top (X^\top R X)^{-1} x_i \cdot p_i^\star (1 - p_i^\star)} q_i^\star = \frac{(X^\top R X)^{-1} x_i}{1 - x_i^\top (X^\top R X)^{-1} x_i \cdot p_i^\star (1 - p_i^\star)} (1 - p_i^\star) \tag{21}$$

The above formula estimates the change in the parameter vector itself. To estimate the change in prediction at a given example x, we take the inner product of the above expression with vector x to get the formula in Definition 3.1.

The approximation here is in assuming the updates converge in one step. Prior works (Koh et al., 2019) quantify the fidelity of such approximation under some assumptions. The effectiveness of TRAK across a variety of settings suggests that the approximation is accurate in regimes that arise in practice.

Incorporating bias terms. The above derivation is commonly done for the case of standard logistic regression, but it also directly extends to the case where the individual predictions incorporate example-dependent bias terms b_i that are independent of θ. In particular, note that the likelihood function after linearization in Step 1 is given by

$$p(z_i; \theta) = \sigma(-y_i \cdot (\nabla_\theta f(z_i; \theta^\star) \cdot \theta + b_i)) \tag{22}$$

where $\sigma(\cdot)$ is the sigmoid function. Because the Hessian and the gradients of the training loss only depend on θ through $p(z_i; \theta)$, and because b_i's are independent of θ, the computation going from Equation (17) to Equation (18) is not affected. The rest of the derivation also remains identical as the bias terms are already incorporated into $p^\star$ and $q^\star$.

[18] This is used also, for instance, to derive the LOO formulas for standard linear regression.

Generalization to other settings. While our derivations in this paper focus on the case of logistic regression, more generally, TRAK can be easily adapted to any choice of model output function as long as the training loss L is a convex function of the model output f. The corresponding entries in the $\mathbf{Q} = \text{diag}(1 - p_i^\star)$ matrix in Definition 3.1 is then replaced by $\partial L / \partial f(z_i)$. The R matrix and the leverage scores also change accordingly, though we do not include them in our estimator (that said, including them may improve the estimator in settings beyond classification).

However, in general one needs care in choosing an appropriate model output function in order to maximize the performance on the linear datamodeling prediction task. If the chosen model output is not well approximated by a linear function of training examples, then that puts an upper bound on the predictive performance of *any* attribution method in our framework. We discuss appropriate choices of model output functions further in Appendix E.6.

E.2. Random projections preserve gradient flow

In Step 2 of TRAK, we use random projections to reduce the dimension of the gradient vectors. Here, we justify this approximation when our model is trained via gradient descent. Similar analysis has been used prior, e.g., by Malladi et al. (2022).

In the limit of small learning rate, the time-evolution of model output $f(z; \theta)$ under gradient descent (or gradient flow) is captured by the following differential equation (Jacot et al., 2018):

$$\frac{df(z;\theta)}{dt} = \sum_i \frac{\partial L(z_i;\theta)}{\partial f(z_i;\theta)} \cdot (\nabla f(z_i;\theta) \cdot \nabla f(z;\theta)) \approx \sum_i \frac{\partial L(z_i;\theta)}{\partial f(z_i;\theta)} \cdot (g_i \cdot g(z)) \tag{23}$$

where g_i and $g(z)$ are the gradients of the final model corresponding to examples z_i and z as before. The approximation is due to assuming that the gradients do not change over time.

If we treat the outputs $\{\hat{f}(z_i;\theta)\}_i$ as time-varying variables, then their time evolution is entirely described by the above system of differential equations (one for each i, replacing z with z_i above). Importantly, the above equations only depend on the gradients through their inner products. Hence, as long as we preserve the inner products to sufficient accuracy, the resulting system has approximately the same evolution as the original one. This justifies replacing the gradient features with their random projections.

E.3. Subsampling the training set

In Step 4 of our algorithm, we ensemble the attribution scores over multiple models. As we investigate in Appendix G.2, this significantly improves TRAK's performance. An important design choice is training each model on a different random subset of the training set.

This choice is motivated by the following connection between TRAK scores and empirical influences (Feldman & Zhang, 2020). Recall that we designed TRAK to optimize the linear datamodeling score. As we discuss in Section 2, datamodels can be viewed as an "oracle" for optimizing the same metric. Further, as Ilyas et al. (2022) observes, datamodels can be viewed as a regularized version of empirical influences (Feldman & Zhang, 2020), which are defined as a difference-in-means estimator,

$$\tau(z_j)_i = \mathbb{E}_{S' \sim \mathcal{D}}[f(z_j; \theta^\star(S')) | z_i \in S'] - \mathbb{E}_{S' \sim \mathcal{D}}[f(z_j; \theta^\star(S')) | z_i \notin S'] \tag{24}$$

where $\mathcal{D}$ is the uniform distribution over α-fraction subsets of training set S. Assuming the expectation over α-fraction subsets is identical to that over subsets of one additional element, we can rearrange the above expression as

$$\tau(z_j)_i = \mathbb{E}_{S' \sim \mathcal{D}}[f(z_j; \theta^\star(S' \cup \{z_i\})) - f(z_j; \theta^\star(S'))]. \tag{25}$$

The above expression is simply the expectation of leave-one-out influence over different random subsets. As the estimate from step 3 of our algorithm is specific to a single training set, we need to average over different subsets in order to approximate the above quantity.

In principle, the estimates computed from $\theta^\star(S')$ only apply to the training examples included in the subset S', since the underlying formula (Definition 3.1) concerns examples that were included for the original converged parameter $\theta^\star$. Hence, when averaging over the models, each model should only update the TRAK scores corresponding to examples in S'. However,

we found that the estimates are marginally better when we update the estimates for the entire training set S (i.e., even those that were not trained on).

Generalization across different α's. A possible concern is that we overfit to a particular regime of α used in evaluating with the LDS. In Figure 9, we evaluate TRAK scores (computing using $\alpha = 0.5$) in other regimes and find that they continue to be highly predictive (though with some degradation in correlation). More generally, our various counterfactual evaluations using the full training set (CIFAR-10 brittleness estimates in Figure 6, the CLIP counterfactuals in Figure 4) indicate that TRAK scores remain predictive near the $\alpha = 1$ regime.

E.4. Soft-thresholding

Soft-thresholding is a common denoising method in statistics (Donoho, 1995) for when an underlying signal is known to be sparse. We apply the soft thresholding operator $S(\cdot; \lambda)$ defined for any $\tau \in \mathbb{R}^n$ as:

$$\mathcal{S}(\tau; \lambda) = (\tau_i - \lambda) \cdot \mathbf{1}\{\tau_i > \lambda\} + (\tau_i + \lambda) \cdot \mathbf{1}\{\tau_i < -\lambda\}. \tag{26}$$

We choose the soft threshold parameter λ via cross-validation. That is, given a set of trained models, we first estimate attribution scores (15), then sample a range of values for λ, compute corresponding attribution scores by applying (26), and finally select the value of λ that yields that highest linear datamodeling score (Definition 2.3) on the set of trained models.

E.5. Extending to multi-class classification

In Section 3, we instantiated TRAK for binary classifiers; we now show how to extend TRAK to the multi-class setting. Recall that our key insight in the binary case was to linearize the model output function $f(z; \theta)$ around the optimal parameters $\theta^\star(S)$ (see (9)). Our choice of output function (i.e., the raw logit of the classifier) allowed us to then cast the original (non-convex) learning problem of interest as an instance of binary logistic regression with inputs $\nabla_\theta f(z; \theta^\star)$. That is, we made the approximation

$$\theta^\star(S) \approx \arg\min_\theta \sum_{z_i \in S} \log\left[1 + \exp\left(-y_i \cdot \left(\nabla_\theta f(z_i; \theta^\star)^\top \theta + b_i\right)\right)\right], \tag{27}$$

and then leveraged Definition 3.1.

To apply this same approach to the c-class setting (for $c > 2$), one possibility is to first transform the problem into c^2 binary classification problems, then apply the approach from Section 3.2 directly. (For example, Malladi et al. (2022) use this transformation to apply the neural tangent kernel to c-way classification problems.) In large-scale settings, however, it is often expensive or infeasible to study of all c^2 subproblems, e.g., ImageNet has $c = 1000$ classes.

We thus take a different approach. In short, we leverage the fact that we always have labels available (even for test examples) to reduce the multi-class classification problem to a *single* logistic regression. More specifically, for an example $z = (x, y)$, we define the model output function

$$f(z; \theta) := \log\left(\frac{p(z; \theta)}{1 - p(z; \theta)}\right), \tag{28}$$

where $p(z; \theta)$ is the softmax probability assigned to the *correct* class.

A crucial property of the model output function (28) is that it allows us to rewrite the loss function for c-way classification as

$$L(z; \theta) = -\log(p(z; \theta)) \tag{29}$$

$$= \log\left[1 + \exp\left(-f(z; \theta)\right)\right], \tag{30}$$

where the first line is the definition of cross-entropy loss, and the second line comes from (28). As a result, if we linearize $f(z; \theta)$ as in Step 1 above (Section 3.2), we can make the approximation

$$\theta^\star(S) \approx \arg\min_\theta \sum_{z_i \in S} \log\left[1 + \exp\left(-\nabla_\theta f(z_i; \theta^\star)^\top \theta + b_i\right)\right].$$

This approximation is identical to the one we made for the binary case (see (27)). We can thus treat the multi-class problem as a single binary logistic regression with inputs $\nabla_\theta f(z_i; \theta^\star)$[19] and then apply Steps 2-5 from Section 3.2 directly to this binary problem.

[19] Note that the corresponding "labels" for this logistic regression are actually identically equal to one—to see this, compare (30) to

E.6. Linearity and model output function

We study linear predictors derived from attribution scores, as linearity is a latent assumption for many popular attribution methods. Linearity also motivates our choices of model output functions.

Latent assumption of linearity. Our evaluation of data attribution methods cast them as linear predictors. While not always immediate, linearity is a latent assumption behind most of the prior methods that we evaluate in this paper. Datamodels and Shapley values satisfy additivity by construction (Ghorbani & Zou, 2019; Jia et al., 2019). The approach based on influence functions (Koh & Liang, 2017; Koh et al., 2019) typically uses the sum of LOO influences to estimate influences for groups of examples. Similarly, empirical (or subsampled) influences (Feldman & Zhang, 2020) also correspond to a first-order Taylor approximation of the model output function. The TracIn estimator also implicitly assumes linearity (Pruthi et al., 2020).

That said, others works also incorporate additional corrections beyond the first order linear terms (Basu et al., 2019) and find the resulting predictions better approximate the true influences.

Choice of model output function f. In our experiments, we choose the model output function suitable for the task at hand: for classification and language modeling, we used a notion of margin that is equivalent to the logit function, while for CLIP, we used a similar one based on the CLIP loss.

Our particular choice of the logit function ($\log p/(1-p)$) in the multi-class classification case was motivated by theoretical (Saunshi et al., 2023) and empirical (Ilyas et al., 2022) observations from prior works. In particular, this choice of model output function is well approximated by *linear* datamodels, both in practice and in theory. A slightly different definition of margin used in Ilyas et al. (2022)—where the margin is computed as the logit for the correct class minus the second highest class—can also be viewed as an approximation to the one used here.

More generally, choosing a good f boils down to linearizing (w.r.t. θ) as much of the model output as possible, but not too much. On one extreme, choosing $f(z) = z$ (i.e., linearizing nothing, as there is no dependence on θ) means that the one-step Newton approximation has to capture all of the non-linearity in both the model and the dependence of L on f; this is essentially the same approximation used by the Hessian-based influence function. On the other extreme, if we choose $f = L$, we linearize too much, which does not work well as L in general is highly non-linear as a function of f.

E.7. The CLIP model output function

The CLIP loss. A CLIP model with parameters θ takes in an image-caption pair (x, y) and outputs an image embedding $\phi(x; \theta)$ and a text embedding $\psi(y; \theta)$. Given a (random) batch of training examples $B = \{(x_1, y_1), ..., (x_n, y_n)\}$, the CLIP training loss computes all $n \times n$ pairwise cosine similarities between the image and text embeddings

$$S_{ij} \coloneqq \phi(x_i; \theta) \cdot \psi(y_j; \theta),$$

and aims to maximize the cosine similarities S_{ii} of correct pairs while minimizing the cosine similarities S_{ij}, for $i \neq j$, of incorrect pairs. More specifically, the training loss of example $(x_i, y_i) \in B$ is defined as the following symmetric cross entropy over the similarity scores S_{ij}:

$$L(x_i, y_i; \theta) = -\log \frac{\exp(S_{ii})}{\sum_{1 \leq j \leq n} \exp(S_{ij})} - \log \frac{\exp(S_{ii})}{\sum_{1 \leq j \leq n} \exp(S_{ji})}, \tag{31}$$

where the first term corresponds to matching each image x_i to its correct caption y_i, and the second term corresponds to matching each caption to its correct image. In effect, we are solving two classification problems: one where the images are inputs and captions (from the same batch) are labels, and vice versa.

Reducing to classification. Recall that in the classification setting we trained the model with the cross entropy loss (i.e., $-\log p(z; \theta)$, where $p(z; \theta)$ is the correct-class probability), and used the model output function $f(z; \theta) = \log p(z; \theta)/(1 - p(z; \theta))$ (Equation (28)), i.e., the logit transform of the correct-class probability to compute TRAK scores.

(27). This does not change the resulting attributions, however, as Definition 3.1 only depends on labels through its dependence on the correct-class probability p_i^*.

To take advantage of the same formula in the CLIP setting, note that our loss (31) can be viewed as having the form

$$L(x_i, y_i; \theta) = -\log p_1(x_i, y_i; \theta) - \log p_2(x_i, y_i; \theta),$$

where $p_1(x_i, y_i; \theta)$ corresponds to the probability of matching an image to its corresponding caption based on the cosine similarity, and likewise for $p_2(x_i, y_i; \theta)$. A natural choice of model output function in this case, then, is using the sum of the model output functions corresponding to the two classification problems:

$$\begin{aligned} f(x_i, y_i; \theta) &:= \log\left(\frac{p_1(x_i, y_i; \theta)}{1 - p_1(x_i, y_i; \theta)}\right) + \log\left(\frac{p_2(x_i, y_i; \theta)}{1 - p_2(x_i, y_i; \theta)}\right) \\ &= -\log \sum_{1 \le j \le n} \exp(S_{ij} - S_{ii}) - \log \sum_{1 \le j \le n} \exp(S_{ji} - S_{ii}). \end{aligned}$$

Indeed, this choice allows us once again (see Appendix E.5) to reduce our problem to an instance of logistic regression and apply the same formula for influence approximation (Definition 3.1) as before.

F. Additional Results

F.1. Full LDS evaluation

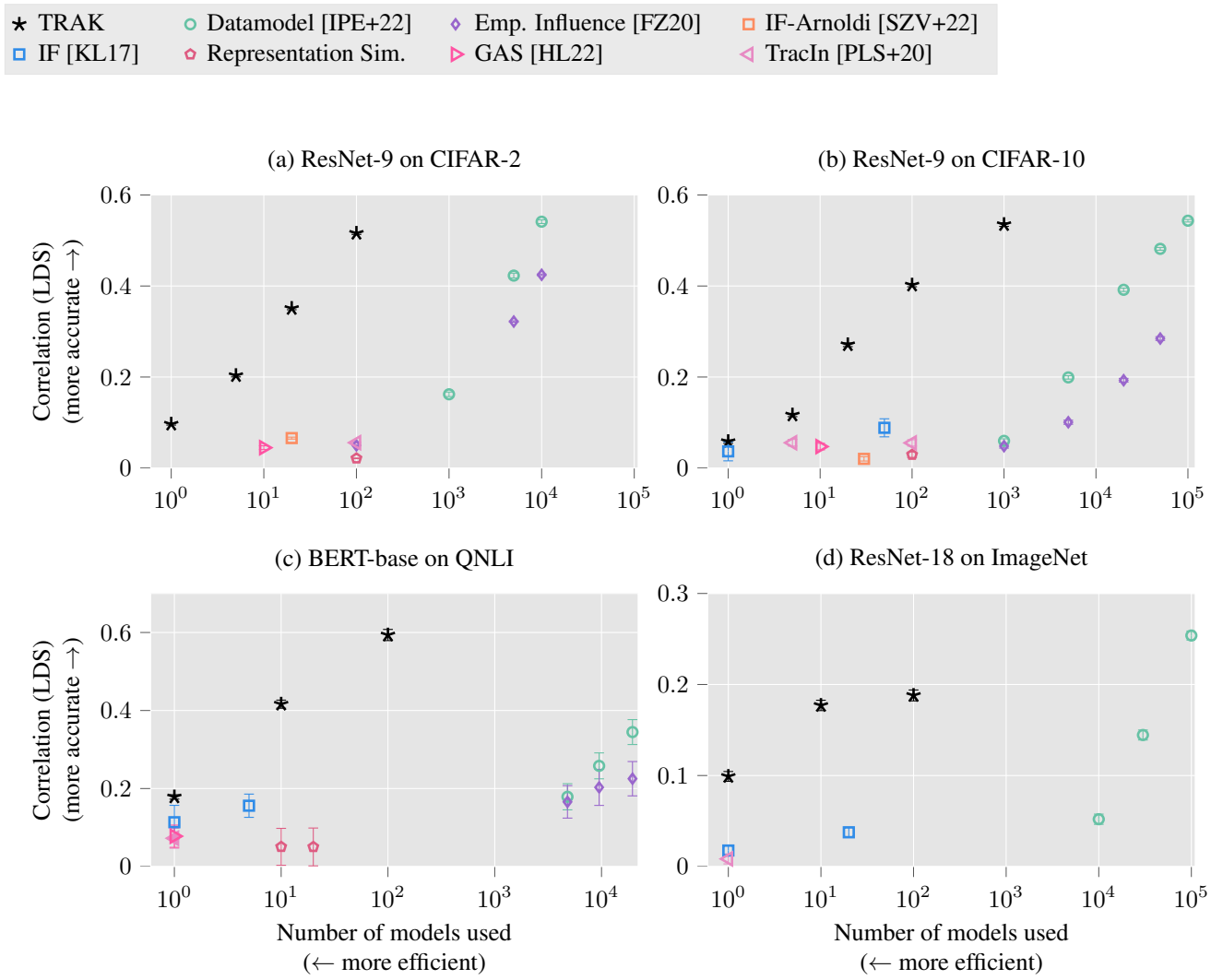

Figure 8. TRAK *achieves state-of-the-art tradeoffs between attribution efficacy and efficiency.* We use TRAK to attribute ResNet-9 classifiers trained on CIFAR-2 and CIFAR-10; ResNet-18 classifiers trained on ImageNet; and BERT-base models finetuned on QNLI. The x-axis indicates the computational cost measured as the number of trained models that a given method uses to compute attribution scores. The y-axis indicates the method's efficacy as measured by the linear datamodeling score (LDS). Error bars indicate 95% bootstrap confidence intervals.

Generalization across α's. In Figure 9 left, we compare the linear datamodeling scores (LDS) evaluated on $\alpha = 0.5$ sub-sampled training sets to those evaluated on $\alpha = 0.75$. (The numbers are overall lower as these are evaluated on data where only one model was trained on each subset,instead of averaging over 5 models; hence, there is more noise in the data.) As we observe, the LDS scores on different α's are highly correlated, suggesting that TRAK scores computed on a single α generalize well.

LDS correlation between TRAK and datamodels. In Figure 9 right, we compare the LDS correlations of datamodels to that of TRAK and find that they are correlated across examples; in general, TRAK also performs better on examples on which datamodels perform better.

Dataset		TRAK	TracIn	Influence function	Datamodels
CIFAR-2	# models	5	100	-	1,000
	Time (min.)	3	100	-	500
	LDS	**0.203(3)**	0.056(2)	-	0.162(5)
CIFAR-10	# models	20	20	1	5,000
	Time (min.)	20	60	20,000	2,500
	LDS	**0.271(4)**	0.056(7)	0.037(13)	0.199(4)
QNLI	# models	10	1	1	20,000
	Time (min.)	640	284	18,000	176,000
	LDS	**0.416(10)**	0.077(29)	0.114(43)	0.344(32)
ImageNet	# models	100	1	20	30,000
	Time (min.)	2920	76	>100,000	525,000
	LDS	**0.188(6)**	0.008(6)	0.037(6)	0.1445(6)

Table 2. Comparison of different data attribution methods. We quantify various data attribution methods (TRAK, TracIn (Pruthi et al., 2020), influence function (Koh & Liang, 2017), and datamodels (Ilyas et al., 2022)) in terms of both their *predictiveness*—as measured by the linear datamodeling score—as well as their *computational efficiency*—as measured by either the total computation time (wall-time measured in minutes on a single A100 GPU; see Appendix C.5 for details) or the number of trained models used to compute the attribution scores. The errors indicate 95% bootstrap confidence intervals. Sampling-based methods (datamodels and empirical influences) can outperform TRAK when allowed to use more computation, but this leads to a significant increase in computational cost.

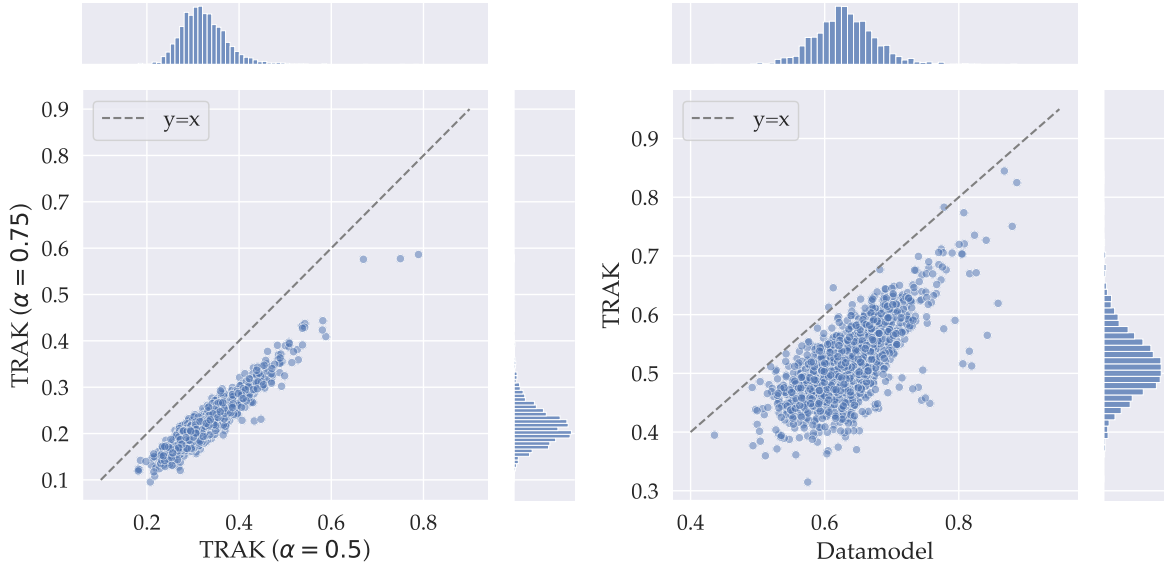

Figure 9. **(Left)** The LDS of CIFAR-2 TRAK scores computed with $\alpha = 0.5$ models then evaluated on either models trained with either $\alpha = 0.5$ or $\alpha = 0.75$. Each point corresponds to a validation example. **(Right)** The LDS of CIFAR-2 datamodel scores compared with that of TRAK. Here, the LDS is measured on two different estimators.

F.2. TRAK examples

Example	Highest TRAK score (+)	Lowest TRAK score (-)
Q: What was a major success, especially in rebuilding Warsaw? **A:** Like many cities in Central and Eastern Europe, infrastructure in Warsaw suffered considerably during its time as an Eastern Bloc economy – though it is worth mentioning that the initial Three-Year Plan to rebuild Poland (especially Warsaw) was a major success, but what followed was very much the opposite. **(Yes)**	**Q:** In 1998, the deal was renewed for what amount over four years? **A:** Television money had also become much more important; the Football League received £6.3 million for a two-year agreement in 1986, but when that deal was renewed in 1988, the price rose to £44 million over four years. **(Yes)**	**Q:** Who was a controversial figure due to a corked-bat incident? **A:** Already a controversial figure in the clubhouse after his corked-bat incident, Sammy's actions alienated much of his once strong fan base as well as the few teammates still on good terms with him, (many teammates grew tired of Sosa playing loud salsa music in the locker room) and possibly tarnished his place in Cubs' lore for years to come. **(No)**
Q: What is the name associated with the eight areas that make up a part of southern California? **A:** Southern California consists of one Combined Statistical Area, eight Metropolitan Statistical Areas, one international metropolitan area, and multiple metropolitan divisions. **(Yes)**	**Q:** Was was the name given to the Alsace provincinal court? **A:** The province had a single provincial court (Landgericht) and a central administration with its seat at Hagenau. **(Yes)**	**Q:** What do six of the questions asses? **A:** For each question on the scale that measures homosexuality there is a corresponding question that measures heterosexuality giving six matching pairs of questions. **(No)**
Q: What words are inscribed on the mace of parliament? **A:** The words There shall be a Scottish Parliament, which are the first words of the Scotland Act, are inscribed around the head of the mace, which has a formal ceremonial role in the meetings of Parliament, reinforcing the authority of the Parliament in its ability to make laws. **(No)**	**Q:** Whose name is on the gate-house fronting School Yard? **A:** His name is borne by the big gate-house in the west range of the cloisters, fronting School Yard, perhaps the most famous image of the school. **(No)**	**Q:** What kind of signs were removed form club Barcelona? **A:** All signs of regional nationalism, including language, flag and other signs of separatism were banned throughout Spain. **(Yes)**
Q: What was the percentage of a female householder with no husband present? **A:** There were 158,349 households, of which 68,511 (43.3%) had children under the age of 18 living in them, 69,284 (43.8%) were opposite-sex married couples living together, 30,547 (19.3%) had a female householder with no husband present, 11,698 (7.4%) had a male householder with no wife present. **(Yes)**	**Q:** What percent of household have children under 18? **A:** There were 46,917 households, out of which 7,835 (16.7%) had children under the age of 18 living in them, 13,092 (27.9%) were opposite-sex married couples living together, 3,510 (7.5%) had a female householder with no husband present, 1,327 (2.8%) had a male householder with no wife present. **(Yes)**	**Q:** Roughly how many same-sex couples were there? **A:** There were 46,917 households, out of which 7,835 (16.7%) had children under the age of 18 living in them, 13,092 (27.9%) were opposite-sex married couples living together, 3,510 (7.5%) had a female householder with no husband present, 1,327 (2.8%) had a male householder with no wife present. **(No)**
Q: What did Warsz own? **A:** In actuality, Warsz was a 12th/13th-century nobleman who owned a village located at the modern-day site of Mariensztat neighbourhood. **(Yes)**	**Q:** What company did Ray Kroc own? **A:** It was founded in 1986 through the donations of Joan B. Kroc, the widow of McDonald's owner Ray Kroc. **(Yes)**	**Q:** What did Cerberus guard? **A:** In Norse mythology, a bloody, four-eyed dog called Garmr guards Helheim. **(No)**
Q: What words are inscribed on the mace of parliament? **A:** The words There shall be a Scottish Parliament, which are the first words of the Scotland Act, are inscribed around the head of the mace, which has a formal ceremonial role in the meetings of Parliament, reinforcing the authority of the Parliament in its ability to make laws. **(No)**	**Q:** Whose name is on the gate-house fronting School Yard? **A:** His name is borne by the big gate-house in the west range of the cloisters, fronting School Yard, perhaps the most famous image of the school. **(No)**	**Q:** What kind of signs were removed form club Barcelona? **A:** All signs of regional nationalism, including language, flag and other signs of separatism were banned throughout Spain. **(Yes)**

Figure 10. Top TRAK *attributions for* QNLI *examples.* Yes/No indicates the label (entailment vs. no entailment).

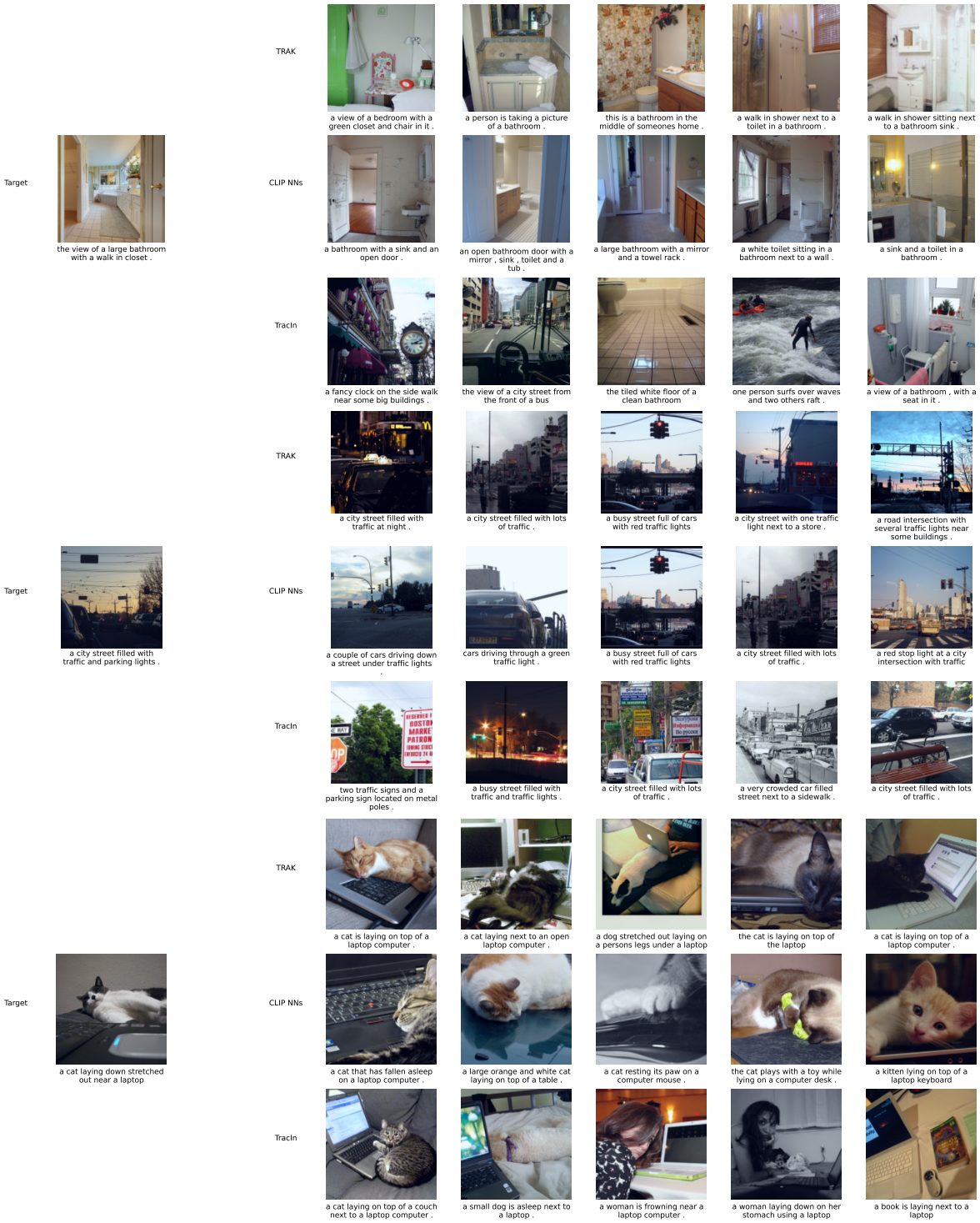

Figure 11. Top attributions for CLIP *models trained on* MS COCO. We display random test examples and their corresponding most helpful (highest-scoring) and most detracting (lowest-scoring) training examples according to TRAK, CLIP similarity distance, and TracIn.

G. Ablation Studies

We perform a number of ablation studies to understand how different components of TRAK affect its performance. Specifically, we study the following:

- The dimension of the random projection, k. Section 3.2).
- The number of models ensembled, M. Section 3.2).
- Proxies for ensembles to further improve TRAK's computational efficiency.
- The role of different terms in the influence estimation formula (Equation (16)).
- Alternative choice of the kernel (using last layer representations).
- Alternative methods of ensembling over models.

As in Section 4, we evaluate the linear datamodeling score (LDS) on models trained on the CIFAR-2, CIFAR-10, and QNLI datasets. Note that the LDS is in some cases lower than the counterparts in Figure 8 as we use a smaller projected dimension (k) and do not use soft-thresholding in these experiments.

G.1. Dimension of the random projection

Recall that when we compute TRAK we reduce the dimensionality of the gradient features using random projections (Step 2 of Section 3.2). Intuitively, as the resulting dimension k increases, the corresponding projection better preserves inner products, but is also more expensive to compute. We now study how the choice of the projection dimension k affects TRAK's attribution performance.

Figure 12 (Left) shows that as we increase the dimension, the LDS initially increases as expected; random projections to a higher dimension preserve the inner product more accurately, providing a better approximation of the gradient features. However, beyond a certain point, increasing projection dimension *decreases* the LDS. We hypothesize that using random projections to a lower dimension has a regularizing effect that competes with the increase in approximation error.[20] Finally, the dimension at which LDS peaks *increases* as we increase the number of models M used to compute TRAK.

G.2. Number of models used in the ensemble

An important component of computing TRAK is ensembling over multiple independently trained models (Step 4 in Section 3.2). In our experiments, we average TRAK's attribution scores over ensembles of size ranging from 1 to 100. Here, we quantify the importance of this procedure on TRAK's performance.

Figure 12 (Right) shows that TRAK enjoys a significantly better data attribution performance with more models. That said, even without ensembling (i.e., using a single model), TRAK still performs better (e.g., LDS of 0.096 on CIFAR-2) than all prior gradient-based methods that we evaluate.

G.3. Proxies for model ensembles in compute-constrained settings

In Appendix G.2 we saw that ensembling leads to significantly higher efficacy (in terms of LDS). In many settings, however, it is computationally expensive to train several independent models to make an ensemble. Hence, we study whether there is a cheaper alternative to training multiple independent models that does not significantly sacrifice efficacy. To this end, we explore two avenues of approximating the full ensembling step while dramatically reducing the time required for model training. In particular, we investigate:

1. using multiple checkpoints from each training trajectory;

[20] Indeed, we can view our approach of first projecting features to a lower dimension and then performing linear regression in the compressed feature space, as an instance of *compressed linear regression* (Maillard & Munos, 2009) and also related to principal components regression (Thanei et al., 2017). These approaches are known to have a regularizing effect, so TRAK may also benefit from that effect.

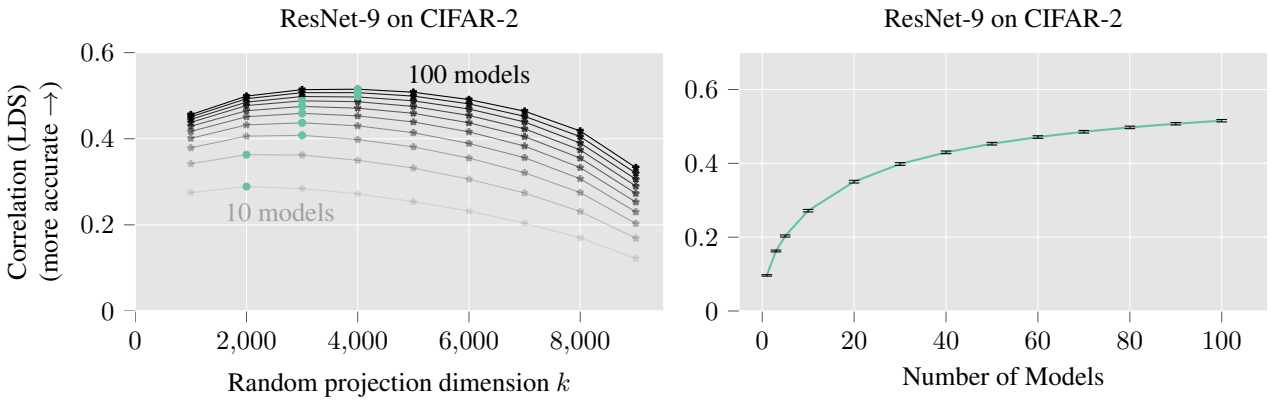

Figure 12. **Left:** *The impact of the dimension of random projection on* TRAK*'s performance on* CIFAR-2. Each line corresponds to a different value of $M \in \{10, 20, ..., 100\}$ (the number of models TRAK is averaged over); darker lines correspond to higher M. As we increase the projected dimension, the LDS initially increases. However, beyond a certain dimension, the LDS begins to decrease. The "optimal" dimension (i.e., the peak in the above graph) increases with higher M. **Right:** *The impact of ensembling more models on* TRAK*'s performance on* CIFAR-2. The performance of TRAK as a function of the number of models used in the ensembling step. TRAK scores are computed with random projections of dimension $k = 4000$.

2. using checkpoints from early training, long before the model has converged.

Multiple checkpoints from each training trajectory. We compute TRAK scores using a *fixed* number of checkpoints, but while varying the number of independently-trained models. For example, for 100 checkpoints, we can use the final checkpoints from 100 independently-trained models, the last two checkpoints from 50 independently-trained models, etc. We observe (see Table 4) that TRAK achieves comparable LDS when we use last T checkpoints along the trajectory of the same models as a proxy for independently-trained models in the ensembling step.

Using checkpoints from early training. We explore whether each of the models in the ensemble has to be fully trained to convergence. In particular, we study the effect of using checkpoints from early epochs on the LDS. While TRAK benefits from using later-epoch gradient features, it maintains its efficacy even when we use gradient features from training runs long before reaching convergence (see Table 3). Leveraing this can further improve the computational efficiency of TRAK.

# training epochs	LDS ($M = 100$)
1	0.100
5	0.204
10	0.265
15	0.293
25	0.308

Table 3. The performance of TRAK on CIFAR-10 as a function of the epoch at which we terminate model training. In all cases, TRAK scores are computed with projection dimension $k = 1000$ and $M = 100$ independently trained models.

# independent models	LDS
5	0.329
6	0.340
10	0.350
100	0.355

Table 4. TRAK maintains its efficacy when we use multiple checkpoints from different epochs of the same training run instead of checkpoints from independently-trained models (CIFAR-10). In all cases, $M = 100$ checkpoints and projection dimension $k = 4000$ are used to compute TRAK scores.

G.4. Role of different terms.

The TRAK estimator (Equation (16)) has a number of different components. We label each component (of the single model estimator) as follows:

$$\tau(z)_i = \frac{\phi(z)^\top \overbrace{(\Phi^\top R \Phi)^{-1}}^{\text{reweighting}} \phi(z_i) \cdot \overbrace{\frac{1}{1+e^{f(z_i)}}}^{\text{loss gradient}}}{1 - \underbrace{h_i}_{\text{leverage score}}}$$

We ablate each of the terms above and re-evaluate the resulting variant of TRAK on CIFAR-2. Our results in Table 5 indicate the following:

- **Reweighting:** Experiment 6 shows that this matrix is a critical part of TRAK's performance. Conceptually, this matrix distinguishes our estimator from prior gradient based similarity metrics such as TracIn.
- **Diagonal term R:** The full reweighting matrix includes a diagonal term R. Although it is theoretically motivated by Definition 3.1, including this term results in lower LDS, so we do not include it (Experiments 2,4).
- **Loss gradient**: This term corresponds to the $\mathbf{Q}$ matrix (Equation (14)) and encodes the probability of the incorrect class, $1 - p_i$; the name is based on the derivation in Appendix E.1, where this term corresponds to scalar associated with the gradient of the loss. Intuitively, this term helps reweight training examples based on on models' confidence on them. Experiment 5 shows that this term improves the performance substantially.
- **Leverage score:** This term does not impact the LDS meaningfully, so we do not include it (Experiments 1,2).
- **Averaging "out" vs "in":** Averaging the estimator and the loss gradient term separately, then re-scaling by the average loss gradient results in higher LDS (Experiment 3).

Experiment	Reweighting	Loss	Diagonal R	Leverage	Averaging	Correlation
0	✓	✓	✗	✗	out	0.499
1	✓	✓	✗	✓	out	0.499
2	✓	✓	✓	✓	out	0.430
3	✓	✓	✗	✗	in	0.416
4	✓	✓	✓	✗	out	0.403
5	✓	✗	✗	✗	out	0.391
6	✗	✓	✗	✗	out	0.056

Table 5. Ablating the contribution of each term in the TRAK *estimator.* For these experiments, we use random projections of dimenseion $k = 2000$.

G.5. Choice of the kernel

To understand how the choice of the kernel impacts the performance of TRAK, we also compute a version of TRAK using feature representations of the penultimate layer in place of the projected gradients. This choice is equivalent to restricting the gradient features to those of the last linear layer. As Table 6 shows, this method significantly improves on all existing baselines based on gradient approximations,[21] but still underperforms significantly relative to TRAK. This gap suggests that the eNTK is capturing additional information that is not captured by penultimate layer representations. Moreover, the larger gap on CIFAR-10 compared to CIFAR-2 and QNLI (both of which are binary classificaiton tasks) hints that the gap will only widen on more complex tasks.

We note that TRAK applied only to the last layer is almost equivalent to the influence function approximation. Indeed, they perform similarly (e.g., the influence function approximation also achieves a LDS of 0.19 on QNLI).

[21] Note that as with the eNTK, the use of multiple models here is crucial: only using a single model gives a correlation of 0.006.

Dataset	Kernel representation	Linear Datamodeling Score (LDS)
CIFAR-2	eNTK	**0.516**
CIFAR-2	penultimate layer	0.198
CIFAR-10	eNTK	**0.413**
CIFAR-10	penultimate layer	0.120
QNLI	eNTK	**0.589**
QNLI	penultimate layer	0.195

Table 6. Choice of the kernel in TRAK. We compare TRAK computed using the eNTK (i.e., using features derived from full gradients) with TRAK computed using the kernel derived from last layer feature representations. The attribution scores are ensembled over $M = 100$ models.

G.6. Ensembling vs. Averaging the eNTK

There are different ways to ensemble a kernel method given multiple kernels $\{K_i\}_i$: (i) we can average the Gram matrices corresponding to each kernel first and then predict using the averaged kernel (i.e., work with $\overline{K} = \frac{1}{n}\sum K_i$), (ii) we can average their induced features (with respect to some fixed basis of functions) and use the corresponding kernel, or (iii) we can average the predictions derived from each kernel (Atanasov et al., 2023). TRAK's algorithm follows the third approach (Step 4).

Here we ensemble using the first approach instead (i.e., using the averaged eNTK). We do this by first averaging the Gram matrices corresponding to each models' eNTK, using the Cholesky decomposition to extract features from the averaged Gram matrix ($G = LL^\top$), then using resulting features L into the same influence formula (Step 3). We find that computing TRAK with this average eNTK gives a significantly underperforming estimator (LDS of 0.120 on CIFAR-2) than averaging *after* computing the estimator from each eNTK (LDS of 0.499). This gap suggests that the underlying model is better approximated as an ensemble of kernel predictors rather than a predictor based on a single kernel.

G.7. Summary

To summarize the results of our ablation, TRAK performs best when averaging over a sufficient number of models (though computationally cheaper alternatives also work); gradients computed at later epochs; and random projections to sufficiently high—but not too high—dimension. Using the reweighting matrix in Equation (16), as well as deriving the features from the full model gradient are also both critical to TRAK's predictive performance.

H. Fact Tracing

H.1. The FTRACE-TREX Dataset

The training set of FTRACE-TREX is sourced from the TREX dataset (Elsahar et al., 2018), with each training example excerpted from a DBPedia abstract (Hellmann et al., 2013) and annotated with a list of facts it expresses.[22] The test set of FTRACE-TREX is sourced from the LAMA dataset (Petroni et al., 2019), and each test example is a sentence that expresses a single fact—every training example that expresses the same fact is called a "proponent" of this test example. Now, given a test example expressing some fact, the goal of fact tracing (as defined by the FTRACE-TREX benchmark) is to correctly identify the corresponding proponents from the training set.

More precisely, Akyurek et al. (2022) propose the following evaluation methodology, which we follow exactly (with the exception that, due to computational constraints, we use a smaller 300M-parameter `mt5-small` model instead of the 580M-parameter `mt5-base`). We first finetune the pretrained language model (Raffel et al., 2020) on the training set of FTRACE-TREX. Then, we iterate through the FTRACE-TREX test set and find the examples on which the pre-trained model is incorrect and the finetuned model is correct,[23] which Akyurek et al. (2022) refer to as the "novel facts" learned by the model after finetuning. For each novel fact identified, we collect a set of candidate training examples, comprising all proponents as well as 300 "distractors" from the training set. Akyurek et al. (2022) propose to evaluate different attribution methods based on how well they identify the ground-truth proponents among each candidate set.

Concretely, given an attribution method $\tau(\cdot)$, we compute attribution scores $\tau(z)$ for each of the novel facts in the test set. For each novel fact, we sort the corresponding candidate examples by their score $\tau(z)_i$. Finally, we compute the mean reciprocal rank (MRR), a standard information retrieval metric, of ground-truth proponents across the set of novel facts, defined as

$$\text{MRR} = \sum_{z \in \substack{\text{novel} \\ \text{facts}}} \frac{1}{\min_{i \in \text{proponents}(z)} \text{rank}(\tau(z), i)}.$$

H.2. Fine-tuning details

We finetune the pre-trained language model using the masked language modeling objective (Devlin et al., 2019). In particular, for each training example $z_i \in [K]^L$ (where K is the vocabulary size and L is the maximum passage length), we mask out a subject or object within the passage. (E.g., a training example "Paris is the capital of France" might become an input-label pair ["__ is the capital of France", "Paris"]). We then treat the language modeling problem as multiple separate K-way classification tasks. Each task corresponds to predicting a single token of the masked-out text, given (as input) the entire passage minus the token being predicted. The loss function is the average cross-entropy loss on this sequence of classification tasks.

H.3. Computing TRAK for masked language modeling

The model output function we use, more precisely, is given by:

$$f(z; \theta) = \sum_{j \in \substack{\text{masked} \\ \text{tokens}}} \log \left(\frac{p(z^j | z^{-j}; \theta)}{1 - p(z^j | z^{-j}; \theta)} \right).$$

In particular, to compute this model output function, we compute the model output function (28) for each one of the V-way classification problems separately, then define our model output function as the sum of these computed outputs.

H.4. Counterfactual experiment setup

To understand the possible roots of TRAK's underperformance relative to BM25 on FTRACE-TREX, we carry out a counterfactual analysis. Specifically, for a subset of the FTRACE-TREX test set, we create three corresponding *counterfactual training sets*. Each training set corresponds to removing one of three collections of examples from the FTRACE-TREX

[22] See (Akyurek et al., 2022) for more details on the annotation methodology.

[23] To decide whether a model is "correct" on a given test example, we use MT5 as a conditional generation model. That is, we feed in a masked version of the query, e.g., "__ is the capital of France," and mark the model as "correct" if the conditional generation matches the masked word.

training set:

(a) the union (across all 50 selected novel facts) of the 500 most important training examples for each novel fact, as identified by TRAK (this corresponds to removing $17,914$ total training examples, leaving $1,542,539$ remaining);

(b) the union of the 500 most important training examples for each novel fact, as identified by BM25 ($18,146$ total examples removed, and $1,542,307$ remaining);

(c) the union of the proponents—as defined by FTRACE-TREX—for each novel fact ($10,780$ examples removed, and $1,549,673$ remaining)

Then, starting from a pre-trained `mt5-small` model (the same model that we finetuned in (B) above to identify novel facts), we finetune several models on each counterfactual training set, and compute their average accuracy on the selected subset of 50 novel facts. Note that, by construction, we know that on this subset (i) the pre-trained model has an accuracy of 0%; and (ii) finetuning on the entire FTRACE-TREX training set (i.e., with no examples removed) yields models with 100% accuracy.[24] As for the counterfactual training sets, one should note that:

- Counterfactual training set (c) is missing all of the proponents for our subset of 50 novel facts—we would thus expect the corresponding finetuned model to have very low accuracy. In particular, there is ostensibly no direct evidence for *any* of the novel facts of interest anywhere in this counterfactual training set.
- Being constructed with BM25, counterfactual training set (b) has high lexical overlap with the novel facts of interest. Since BM25 performs well on the FTRACE-TREX benchmark, we would also expect the resulting models to have low accuracy.

In Figure 5, we report the resulting models' average performance on the set of 50 selected novel facts. What we find is that, counter to the above intuition, *only the* TRAK*-based counterfactual training set is able to significantly change model behavior*. That is, the counterfactual effect of removing the most important images as identified by TRAK on the selected subset of novel facts is significantly higher than both (a) that of removing the most important images according to BM25; and (b) that of removing the *ground-truth proponents* of the facts as indicated by the FTRACE-TREX benchmark.

H.5. Potential explanations for counterfactual results

We discuss some potential reasons for why TRAK outperforms the FTRACE-TREX ground-truth in our counterfactual evaluation (some of which Akyurek et al. (2022) already discuss in their work):

- There may be errors in the FTRACE-TREX benchmark. (Although, given the drastic difference between the TRAK scores and the ground-truth labels in their ability to identify counterfactually important abstracts, such data errors are unlikely to be the sole culprit.)
- Models may be answering queries by *combining* facts from the training set. For example, neither "The largest pyramid is in Giza" nor "Giza is a city in Egypt" would be ground-truth proponents for the query "Which country is home to the largest pyramid?" in FTRACE-TREX, but a model that learns both of these facts may still be able to correctly answer that query.
- Alternatively, models may be learning from the syntactic rather than semantic structure of abstracts. For example, a model may correctly answer that a person from Korea is called a "Korean" by learning from an abstract which says "A person from Bulgaria is Bulgarian."

[24] In particular, recall that in order for a test example to be categorized as a "novel fact," it must be both (a) incorrectly handled by the pre-trained `mt5-small` model and (b) correctly handled by a finetuned model.

I. Future Work

I.1. Further applications of TRAK

Prior works have demonstrated the potential of leveraging data attribution for a variety of downstream applications, ranging from explaining predictions (Koh & Liang, 2017; Kong et al., 2022), cleaning datasets (Jia et al., 2019), removing poisoned examples (Lin et al., 2022) to quantifying uncertainty (Alaa & Van Der Schaar, 2020). Given the effectiveness of TRAK, we expect that using it in place of existing attribution methods will improve the performance in many of these downstream applications. Moreover, given its computational efficiency, TRAK can expand the settings in which these prior data attribution methods are feasible. Indeed, we already saw some examples in Appendix A. We highlight a few promising directions in particular:

Fact tracing and attribution for generative models. Fact tracing, which we studied in Section 5.2, is a problem of increasing relevancy as large language models are widely deployed. Leveraging TRAK for fact tracing, or attribution more broadly, may help understand the capabilities or improve the trustworthiness of recent models such as GPT-3 (Brown et al., 2020) and ChatGPT,[25] by tracing their outputs back to sources in a way that is faithful to the actual model. More broadly, attribution for generative models (e.g., stable diffusion (Ho et al., 2020; Rombach et al., 2022)) is an interesting direction for future work.

Optimizing datasets. TRAK scores allow one to quantify the impact of individual training examples on model predictions on a given target example. By aggregating this information, we can optimize what data we train the models on, for instance, to choose *coresets* or to select new data for *active learning*. Given the trend of training models on ever increasing size of datasets (Hoffmann et al., 2022), filtering data based on their TRAK scores can also help models achieve with the benefits of scale without the computational cost.

Another advantage of TRAK is that it is fully differentiable in the input (note that the associated gradients are different from the gradients with respect to model parameters that we use when computing TRAK). One potential direction is to leverage this differentiability for *dataset distillation*. Given the effectiveness of the NTK for this problem (Nguyen et al., 2021a), there is potential in leveraging TRAK—which uses the eNTK—in this setting.

I.2. Understanding and improving the TRAK estimator

Empirical NTK. TRAK leverages the empirical NTK to approximate the original model. Better understanding of when this approximation is accurate may give insights into improving TRAK's efficacy. For example, incorporating higher order approximations (Huang & Yau, 2020; Bai & Lee, 2020) beyond the linear approximation used in TRAK is a possible direction.

Training dynamics and optimization. Prior works (Leclerc & Madry, 2020; Lewkowycz et al., 2020) suggest that neural network training can exhibit two stages or regimes: in the first stage, the features learned by the network evolve rapidly; in the second stage, the features remain approximately invariant and the overall optimization trajectory is more akin a convex setting. We can view our use of the final eNTK as modeling this second stage. Understanding the extent to which the first stage (which TRAK does not model) accounts for the remaining gap between true model outputs and TRAK's predictions may help us understand the limits of our method as well as improve its efficacy. Another direction is to study whether properly accounting for other optimization components used during training, such as mini-batches, momentum, or weight decay, can improve our estimator.

Ensembles. As we saw in Appendix G.2, computing TRAK over an ensemble of models significantly improves its efficacy. In particular, our results suggest that the eNTK's derived from independently trained models capture non-overlapping information. Better understanding of the role of ensembling here may us better understand the mechanisms underlying ensembles in other contexts and can also provide practical insights for improving TRAK's efficiency. For instance, understanding when model checkpoints from a single trajectory can approximate the full ensemble (Appendix G.3) can be valuable in settings where it is expensive to even finetune several models.

[25] https://chat.openai.com/