Research Article

Influence Functions for Scalable Data Attribution in Diffusion Models

Bruno Mlodozeniec[1,2], Runa Eschenhagen[1], Juhan Bae[3,4], Alexander Immer[2,5], David Krueger[6], Richard Turner[1,7]

1. Department of Engineering, University of Cambridge, United Kingdom; 2. Max Planck Institute for Intelligent Systems, Stuttgart, Germany; 3. Department of Computer Science, University of Toronto, Canada; 4. Vector Institute, Toronto, Canada; 5. Department of Computer Science, ETH Zürich, Zurich, Switzerland; 6. Mila - Quebec Artificial Intelligence Institute, Montreal, Canada; 7. The Alan Turing Institute, London, United Kingdom

Diffusion models have led to significant advancements in generative modelling. Yet their widespread adoption poses challenges regarding data attribution and interpretability. In this paper, we aim to help address such challenges in diffusion models by developing an *influence function* framework. Influence function-based data attribution methods approximate how a model's output would have changed if some training data were removed. In supervised learning, this is usually used for predicting how the loss on a particular example would change. For diffusion models, we focus on predicting the change in the probability of generating a particular example via several proxy measurements. We show how to formulate influence functions for such quantities and how previously proposed methods can be interpreted as particular design choices in our framework. To ensure scalability of the Hessian computations in influence functions, we systematically develop K-FAC approximations based on generalised Gauss-Newton matrices specifically tailored to diffusion models. We recast previously proposed methods as specific design choices in our framework, and show that our recommended method outperforms previous data attribution approaches on common evaluations, such as the Linear Data-modelling Score (LDS) or retraining without top influences, without the need for method-specific hyperparameter tuning.

Corresponding author: Bruno Mlodozeniec, bkm28@cam.ac.uk

1. Introduction

Generative modelling for continuous data modalities — like images, video, and audio — has advanced rapidly propelled by improvements in diffusion-based approaches. Many companies now offer easy access to AI-generated bespoke image content. However, the use of these models for commercial purposes creates a need for understanding how the training data influences their outputs. In cases where the model's outputs are

undesirable, it is useful to be able to identify, and possibly remove, the training data instances responsible for those outputs. Furthermore, as copyrighted works often make up a significant part of the training corpora of these models[1], concerns about the extent to which individual copyright owners' works influence the generated samples arise. Some already characterise what these companies offer as "copyright infringement as a service"[2], which has caused a flurry of high-profile lawsuits[2][3]. This motivates exploring tools for data attribution that might be able to quantify how each group of training data points influences the models' outputs. Influence functions[4][5] offer precisely such a tool. By approximating the answer to the question, "If the model was trained with some of the data excluded, what would its output be?", they can help finding data points most responsible for a low loss on an example, or a high probability of generating a particular example. However, they have yet to be scalably adapted to the general diffusion modelling setting.

Influence functions work by locally approximating how the loss landscape would change if some of the training data points were down-weighted in the training loss (illustrated in Figure 5). Consequently, this enables prediction for how the (local) optimum of the training loss would change, and how that change in the parameters would affect a measurement of interest (e.g., loss on a particular example). By extrapolating this prediction, one can estimate what would happen if the data points were fully removed from the training set. However, to locally approximate the shape of the loss landscape, influence functions require computing and inverting the Hessian of the training loss, which is computationally expensive. One common approximation of the training loss's Hessian is the generalised Gauss-Newton matrix (GGN),[6][7]. The GGN has not been clearly formulated for the diffusion modelling objective before and cannot be uniquely determined based on its general definition. Moreover, to compute and store a GGN for large neural networks further approximations are necessary. We propose using Kronecker-Factored Approximate Curvature (K-FAC),[8][9] to approximate the GGN. It is not commonly known how to apply it to neural network architectures used in diffusion models; for example,[10] resort to alternative Hessian approximation methods because "[K-FAC] might not be applicable to general deep neural network models as it highly depends on the model architecture". However, based on recent work, it is indeed clear that it can be applied to architectures used in diffusion models[11][12], which typically combine linear layers, convolutions, and attention[13].

In this work, we describe a scalable approach to influence function-based approximations for data attribution in diffusion models, using a K-FAC approximation of GGNs as Hessian approximations. We articulate a design space based on influence functions, unify previous methods for data attribution in diffusion models[14] [15] through our framework, and argue for the design choices that distinguish our method from previous ones. One important design choice is the GGN used as the Hessian approximation. We formulate different GGN matrices for the diffusion modelling objective and discuss their implicit assumptions. We empirically ablate variations of the GGN and other design choices in our framework and show that our proposed method

outperforms the existing data attribution methods for diffusion models as measured by common data attribution metrics like the Linear Data-modelling Score[16] or retraining without top influences. Finally, we also discuss interesting empirical observations that challenge our current understanding of influence functions in the context of diffusion models.

Figure 1. Most influential training data points as identified by K-FAC Influence Functions for samples generated by a denoising diffusion probabilistic model trained on CIFAR-10. The top influences are those whose omission from the training set is predicted to most increase the loss of the generated sample. Negative influences are those predicted to most decrease the loss, and the most neutral are those that should change the loss the least.

2. Background

This section introduces the general concepts of diffusion models, influence functions, and the GGN.

2.1. Diffusion Models

bordercolor=orange!50bordercolor=orange!50todo: bordercolor=orange!50BM: Notation: change $x^{(t),x^{(0:t)}}$ to $x_t, x_{0:t}$ for space/better formatting?

Diffusion models are a class of probabilistic generative models that fit a model $p_\theta(x)$ parameterised by parameters $\theta \in \mathbb{R}^{d_{\text{param}}}$ to approximate a training data distribution $q(x)$, with the primary aim being to sample new data $x \sim p_\theta(\cdot)$[17][13][18]. This is usually done by augmenting the original data x with T fidelity levels as $x^{(0:T)} = [x^{(0)}, \dots, x^{(T)}]$ with an augmentation distribution $q(x^{(0:T)})$ that satisfies the following criteria: 1) the

highest fidelity $x^{(0)}$ equals the original training data $q(x^{(0)}) = q(x)$, 2) the lowest fidelity $x^{(T)}$ has a distribution that is easy to sample from, and 3) predicting a lower fidelity level from the level directly above it is simple to model and learn. To achieve the above goals, q is typically taken to be a first-order Gaussian auto-regressive (diffusion) process: $q(x^{(t)}|x^{(0:t-1)}) = \mathcal{N}(x^{(t)}|\lambda_t x^{(t-1)}, (1-\lambda_t)^2 I)$, with hyperparameters λ_t set so that the law of $x^{(T)}$ approximately matches a standard Gaussian distribution $\mathcal{N}(0, I)$. In that case, the reverse conditionals $q(x^{(t-1)}|x^{(t:T)}) = q(x^{(t-1)}|x^{(t)})$ are first-order Markov, and if the number of fidelity levels T is high enough, they can be well approximated by a diagonal Gaussian, allowing them to be modelled with a parametric model with a simple likelihood function, hence satisfying (3) [18]. The marginals $q(x^{(t)}|x^{(0)}) = \mathcal{N}(x^{(t)}|\prod_{t'=1}^{t}\lambda_t x^{(0)}, (1-\prod_{t'=1}^{t}\lambda_{t'}^2)I)$ also have a simple Gaussian form, allowing for the augmented samples to be sampled as:

$$x^{(t)} = \prod_{t'=1}^{t}\lambda_t x^{(0)} + (1-\prod_{t'=1}^{t}\lambda_{t'}^2)^{1/2}\epsilon^{(t)}, \quad \text{with } \epsilon^{(t)} \sim \mathcal{N}(0, I). \tag{1}$$

Diffusion models are trained to approximate the reverse conditionals $p_\theta(x^{(t-1)}|x^{(t)}) \approx q(x^{(t-1)}|x^{(t)})$ by maximising log-probabilities of samples $x^{(t-1)}$ conditioned on $x^{(t)}$, for all timesteps $t = 1, \ldots, T$. We can note that $q(x^{(t-1)}|x^{(t)}, x^{(0)})$ has a Gaussian distribution with mean given by:

$$\mu_{t-1|t,0}(x^{(t)}, \epsilon^{(t)}) = \frac{1}{\lambda_t}\left(x^{(t)} - \frac{1-\lambda_t^2}{(1-\prod_{t'=1}^{t}\lambda_{t'}^2)^{1/2}\epsilon^{(t)}}\right), \quad \text{with-}\epsilon^{(t)} \stackrel{\text{def}}{=} \frac{(x^{(t)} - \prod_{t'=1}^{t}\lambda_{t'}x^{(0)})}{(1-\prod_{t'=1}^{t}\lambda_{t'}^2)^{1/2}}$$

as in Equation (1). In other words, the mean is a mixture of the sample $x^{(t)}$ and the noise $\epsilon^{(t)}$ that was applied to $x^{(0)}$ to produce it. Hence, we can choose to analogously parameterise $p_\theta(x^{t-1}|x^{(t)})$ as $\mathcal{N}(x^{t-1}|\mu_{t-1|t,0}(x^{(t)}, \epsilon_\theta^t(x^{(t)})), \sigma_t^2 I)$. That way, the model $\epsilon_\theta^{(t)}(x^{(t)})$ simply predicts the noise $\epsilon^{(t)}$ that was added to the data to produce $x^{(t)}$. The variances σ_t^2 are usually chosen as hyperparameters[13]. With that parameterisation, the negative expected log-likelihood $\mathbb{E}_{q(x^{t-1},x^{(t)}|x^{(0)})}[-\log p(x^{t-1}|x^{(t)})]$, up to scale and shift independent of θ or $x^{(0)}$, can be written as[13][18]:[1]

$$\ell_t(\theta, x^{(0)}) = \mathbb{E}_{\epsilon^{(t)},x^{(t)}}[\|\epsilon^{(t)} - \epsilon_\theta^t(x^{(t)})\|^2] \qquad \begin{array}{c}\epsilon^{(t)} \sim \mathcal{N}(0, I) \\ x^{(t)} = \prod_{t'=1}^{t}\lambda_t x^{(0)} + (1-\prod_{t'=1}^{t}\lambda_{t'}^2)^{1/2}\epsilon^{(t)}\end{array} \tag{2}$$

This leads to a training loss ℓ for the diffusion model $\epsilon_\theta^t(x^{(t)})$ that is a sum of per-diffusion timestep training losses:[2]

$$\ell(\theta, x) = \mathbb{E}_{\tilde{t}}[\ell_{\tilde{t}}(\theta, x)] \qquad \tilde{t} \sim \text{Uniform}([T]).$$

The parameters are then optimised to minimise the loss averaged over a training dataset $\mathcal{D} = \{x_n\}_{n=1}^N$:

$$\theta^*(\mathcal{D}) = \arg\min_\theta \mathcal{L}_\mathcal{D}(\theta) \qquad \mathcal{L}_\mathcal{D}(\theta) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{n=1}^{N}\ell(\theta, x_n). \tag{3}$$

Other interpretations of the above procedure exist in the literature[19][20][21][22].

2.2. *Influence Functions*

The aim of influence functions is to answer questions of the sort "how would my model behave were it trained on the training dataset with some datapoints removed". To do so, they approximate the change in the optimal model parameters in Equation (3) when some training examples $(x_j)_{j\in\mathcal{I}}$, $\mathcal{I} = \{i_1, \dots, i_M\} \subseteq [N]$, are removed from the dataset $\mathcal{D}$. To arrive at a tractable approximation, it is useful to consider a continuous relaxation of this question: how would the optimum change were the training examples $(x_j)_{j\in\mathcal{I}}$ down-weighted by $\varepsilon \in \mathbb{R}$ in the training loss:

$$r_{-\mathcal{I}}(\varepsilon) = \arg\min_{\theta} \frac{1}{N}\sum_{n=1}^{N} \ell(\theta, x_n) - \varepsilon \sum_{j\in\mathcal{I}} \ell(\theta, x_j) \tag{4}$$

The function $r_{-\mathcal{I}} : \mathbb{R} \to \mathbb{R}^{d_{\texttt{param}}}$ (well-defined if the optimum is unique) is the *response function*. Setting ε to $1/N$ recovers the minimum of the original objective in Eq. (3) with examples $(x_{i_1}, \dots, x_{i_M})$ removed.

Under suitable assumptions (see Appendix A), by the Implicit Function Theorem[23], the response function is continuous and differentiable at $\varepsilon = 0$. *Influence functions* can be defined as a linear approximation to the response function $r_{-\mathcal{I}}$ by a first-order Taylor expansion around $\varepsilon = 0$:

$$\begin{aligned} r_{-\mathcal{I}}(\varepsilon) &= r_{-\mathcal{I}}(0) + \left.\frac{dr_{-\mathcal{I}}(\varepsilon')}{d\varepsilon'}\right|_{\varepsilon'=0} \varepsilon + o(\varepsilon) \\ &= \theta^*(\mathcal{D}) + \sum_{j\in\mathcal{I}} \left(\nabla^2_{\theta^*}\mathcal{L}_{\mathcal{D}}(\theta^*)\right)^{-1} \nabla_{\theta^*}\ell(\theta^*, x_j)\varepsilon + o(\varepsilon), \end{aligned} \tag{5}$$

as $\varepsilon \to 0$. See Appendix A for a formal derivation and conditions. The optimal parameters with examples $(x_i)_{i\in\mathcal{I}}$ removed can be approximated by setting ε to $1/N$ and dropping the $o(\varepsilon)$ terms.

Usually, we are not directly interested in the change in parameters in response to removing some data, but rather the change in some *measurement* function $m(\theta^*(\mathcal{D}), x')$ at a particular test input x' (e.g. per-example test loss). We can further make a first-order Taylor approximation to $m(\cdot, x')$ at $\theta^*(\mathcal{D})$ — $m(\theta, x') = m(\theta^*, x') + \nabla^\top_{\theta^*} m(\theta^*, x')(\theta - \theta^*) + o(\|\theta - \theta^*\|_2)$ — and combine it with Eq. (5) to get a simple linear estimate of the change in the measurement function:

$$m(r_{-\mathcal{I}}(\varepsilon), x') = m(\theta^*, x') + \sum_{j\in\mathcal{I}} \nabla^\top_{\theta^*} m(\theta^*, x') \left(\nabla^2_{\theta^*}\mathcal{L}_{\mathcal{D}}(\theta^*)\right)^{-1} \nabla_{\theta^*}\ell(\theta^*, x_j)\varepsilon + o(\varepsilon). \tag{6}$$

2.2.1. *Generalised Gauss-Newton matrix*

Computing the influence function approximation in Eq. (5) requires inverting the Hessian $\nabla^2_\theta \mathcal{L}_{\mathcal{D}}(\theta) \in \mathbb{R}^{d_{\texttt{param}} \times d_{\texttt{param}}}$. In the context of neural networks, the Hessian itself is generally computationally intractable and approximations are necessary. A common Hessian approximation is the generalised Gauss-Newton matrix (GGN). We will first introduce the GGN in an abstract setting of approximating the Hessian for a

general training loss $\mathcal{L}(\theta) = \mathbb{E}_z[\rho(\theta, z)]$, to make it clear how different variants can be arrived at for diffusion models in the next section.

In general, if we have a function $\rho(\theta, z)$ of the form $h_z \circ f_z(\theta)$, with h_z a convex function, the GGN for an expectation $\mathbb{E}_z[\rho(\theta, z)]$ is defined as

$$\mathrm{GGN}(\theta) = \mathbb{E}_z\left[\nabla_\theta^\top f_z(\theta) \nabla^2_{f_z(\theta)} h_z(f_z(\theta)) \nabla_\theta f_z(\theta)\right],$$

where $\nabla_\theta f_z(\theta)$ is the Jacobian of f_z. Whenever f_z is (locally) linear, the GGN is equal to the Hessian $\mathbb{E}_z[\nabla^2_\theta \rho(\theta, z)]$. Therefore, we can consider the GGN as an approximation to the Hessian in which we "linearise" the function f_z. Note that any decomposition of $\rho(\theta, z)$ results in a valid GGN as long as h_z is convex[7].[3] We give two examples below.

Option 1. A typical choice would be for f_z to be the neural network function on a training datapoint z, and for h_z to be the loss function (e.g. ℓ_2-loss), with the expectation $\mathbb{E}_z$ being taken over the empirical (training) data distribution; we call the GGN for this split GGN^{model}. The GGN with this split is exact for linear neural networks (or when the model has zero residuals on the training data)[7].

$$\begin{matrix} f_z := \text{mapping from parameters to model output} \\ h_z := \text{loss function (e.g.-}\ell_2\text{-loss)} \end{matrix} \qquad \rightarrow GGN^{\text{model}}(\theta) \tag{7}$$

Option 2. Alternatively, a different GGN can be defined by using a trivial split of the loss $\rho(\theta, z)$ into the identity map $h_z := id$ and the loss $f_z := \rho(\cdot, z)$, and again taking the expectation over the empirical data distribution. With this split, the resulting GGN is

$$\begin{matrix} f_z := \rho(\cdot, z) \\ h_z := id \end{matrix} \qquad \rightarrow GGN^{\text{loss}}(\theta) = \mathbb{E}_z\left[\nabla_\theta \rho(\theta, z) \nabla_\theta^\top \rho(\theta, z)\right]. \tag{8}$$

This is also called the empirical Fisher[24]. Note that GGN^{loss} is only equal to the Hessian under the arguably more stringent condition that $\rho(\cdot, z)$ — the composition of the model and the loss function — is linear. This is in contrast to GGN^{model}, for which only the mapping from the parameters to the model output needs to be (locally) linear. Hence, we might prefer to use GGN^{model} for Hessian approximation whenever we have a nonlinear loss, which is the case for diffusion models.

3. Scalable influence functions for diffusion models

In this section, we discuss how we adapt influence functions to the diffusion modelling setting in a scalable manner. We also recast data attribution methods for diffusion models proposed in prior work[14][15] as the result of particular design decisions in our framework, and argue for our own choices that distinguish our method from the previous ones.

3.1. Approximating the Hessian

In diffusion models, we want to compute the Hessian of the loss of the form

$$\mathcal{L}_{\mathcal{D}}(\theta) = \mathbb{E}_{x_n}\left[\ell(\theta, x_n)\right] = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})}}\left[\|\epsilon^{(\tilde{t})} - \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\right]\right]\right],$$

where $\mathbb{E}_{x_n}[\cdot] = \left(\frac{1}{N}\sum_{n=1}^{N} \cdot\right)$ is the expectation over the empirical data distribution.[4]

We will describe how to formulate different GGN approximations for this setting.

3.1.1. GGN for diffusion models

Option 1. To arrive at a GGN approximation, as discussed in Section 2.2.1, we can partition the function $\theta \mapsto \|\epsilon^{(t)} - \epsilon_\theta^t(x^{(t)})\|^2$ into the model output $\theta \mapsto \epsilon_\theta^t(x^{(t)})$ and the ℓ_2-loss function $\|\epsilon^{(t)} - \cdot\|^2$. This results in the GGN:

$$\begin{matrix} f_z := \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})}) \\ h_z := \|\ \epsilon^{(\tilde{t})} - \cdot\|^2 \end{matrix} \quad \to \mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})}}\left[\nabla_\theta^\intercal \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\,(2I)\,\nabla_\theta \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right]\right]\right], \tag{9}$$

where I is the identity matrix. This correspond to "linearising" the neural network ϵ_θ^t. For diffusion models, the dimensionality of the output of $\epsilon_\theta^{\tilde{t}}$ is typically very large (e.g. $32 \times 32 \times 3$ for CIFAR), so computing the Jacobians $\nabla_\theta \epsilon_\theta^t$ explicitly is still intractable. However, we can express $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}$ as

$$F_{\mathcal{D}}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x_n^{(\tilde{t})}}\left[\mathbb{E}_{\epsilon_{\mathrm{mod}}}\left[g_n(\theta) g_n(\theta)^\intercal\right]\right]\right]\right], \qquad \epsilon_{\mathrm{mod}} \sim \mathcal{N}\left(\epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})}), I\right) \tag{10}$$

where $g_n(\theta) = \nabla_\theta \|\epsilon_{\mathrm{mod}} - \epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})})\|^2 \in \mathbb{R}^{d_{\mathrm{param}}}$; see Appendix B for the derivation. This formulation lends itself to a Monte Carlo approximation, since we can now compute gradients using auxiliary targets ϵ_{mod} sampled from the model's output distribution, as shown in Equation (10). $F_{\mathcal{D}}$ can be interpreted as a kind of Fisher information matrix[25][7], but it is not the Fisher for the marginal model distribution $p_\theta(x)$.

Option 2. Analogously to Equation (8), we can also consider the trivial decomposition of $\ell(\cdot, x)$ into the identity map and the loss, effectively "linearising" $\ell(\cdot, x)$. The resulting GGN is:

$$\begin{matrix} f_z := \ell(\cdot, x_n) \\ h_z := id \end{matrix} \quad \to \mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}(\theta) = \mathbb{E}_{x_n}\left[\nabla_\theta \ell(\theta, x_n) \nabla_\theta^\top \ell(\theta, x_n)\right], \tag{11}$$

where $\ell(\theta, x)$ is the diffusion training loss defined in Equation (2). This Hessian approximation $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$ turns out to be equivalent to the ones considered in the previous works on data attribution for diffusion models[14][15][10]. In contrast, in this work, we opt for $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}$ in Equation (9), or equivalently $F_{\mathcal{D}}$, since it is arguably a better-motivated approximation of the Hessian than $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$ (c.f. Section 2.2.1).

In[15], the authors explored substituting different (theoretically incorrect) training loss functions into the influence function approximation. In particular, they found that replacing the loss $\|\epsilon^{(t)} - \epsilon_\theta^t(x^{(t)})\|^2$ with the

square norm loss $\|\epsilon_\theta^t(x^{(t)})\|^2$ (effectively replacing the "targets" $\epsilon^{(t)}$ with 0) gave the best results. Note that the targets $\epsilon^{(t)}$ do not appear in the expression for $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}$ in Equation (9).[5]

Hence, in our method substituting different targets would not affect the Hessian approximation. In [15], replacing the targets only makes a difference to the Hessian approximation because they use $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$ (an empirical Fisher) to approximate the Hessian.

3.1.2. K-FAC for diffusion models

While $F_{\mathcal{D}}(\theta)$ and $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$ do not require computing full Jacobians or the Hessian of the neural network model, they involve taking outer products of gradients of size $\mathbb{R}^{d_{\mathrm{param}}}$, which is still intractable. Kronecker-Factored Approximate Curvature K-FAC[8][9] is a common scalable approximation of the GGN to overcome this problem. It approximates the GGN with a block-diagonal matrix, where each block corresponds to one neural network layer and consists of a Kronecker product of two matrices. Due to convenient properties of the Kronecker product, this makes the inversion and multiplication with vectors needed in Equation (6) efficient enough to scale to large networks. K-FAC is defined for linear layers, including linear layers with weight sharing like convolutions[11]. This covers most layer types in the architectures typically used for diffusion models. When weight sharing is used, there are two variants – K-FAC-expand and K-FAC-reduce[12]. For our recommended method, we choose to approximate the Hessian with a K-FAC approximation of $F_{\mathcal{D}}$, akin to[26].

For the parameters θ_l of layer l, the GGN $F_{\mathcal{D}}$ in Equation (10) is approximated by

$$F_{\mathcal{D}}(\theta_l) \approx \frac{1}{N^2} \sum_{n=1}^{N} \mathbb{E}_{\tilde{t}} \left[\mathbb{E}_{x_n^{(\tilde{t})}, \epsilon^{(\tilde{t})}} \left[a_n^{(l)} a_n^{(l)\top} \right] \right] \otimes \sum_{n=1}^{N} \mathbb{E}_{\tilde{t}} \left[\mathbb{E}_{x_n^{(\tilde{t})}, \epsilon^{(\tilde{t})}, \epsilon_{\mathrm{mod}}^{(\tilde{t})}} \left[b_n^{(l)} b_n^{(l)\top} \right] \right], \tag{12}$$

with $a_n^{(l)} \in \mathbb{R}^{d_{\mathrm{in}}^l}$ being the inputs to the lth layer for data point $x_n^{(\tilde{t})}$ and $b_n^{(l)} \in \mathbb{R}^{d_{\mathrm{out}}^l}$ being the gradient of the ℓ_2-loss w.r.t. the output of the lth layer, and $\otimes$ denoting the Kronecker product.[6] The approximation trivially becomes an equality for a single data point and also for deep linear networks with ℓ_2-loss[27][12]. We approximate the expectations in Equation (12) with Monte Carlo samples and use K-FAC-expand whenever weight sharing is used since the problem formulation of diffusion models corresponds to the expand setting in[12]; in the case of convolutional layers this corresponds to[11]. Lastly, to ensure the Hessian approximation is well-conditioned and invertible, we follow standard practice and add a damping term consisting of a small scalar damping factor times the identity matrix. We ablate these design choices in Section 4 (Figures 4, 7 and 9).

3.2. Gradient compression and query batching

In practice, we recommend computing influence function estimates in Equation (6) by first computing and storing the approximate Hessian inverse, and then iteratively computing the preconditioned inner products $\nabla_{\theta^*}^\top m(\theta^*, x) \left(\nabla_{\theta^*}^2 \mathcal{L}_{\mathcal{D}}(\theta^*) \right)^{-1} \nabla_{\theta^*} \ell(\theta^*, x_j)$ for different training datapoints x_j. Following[26], we use query

batching to avoid recomputing the gradients $\nabla_{\theta^*}\ell(\theta^*, x_j)$ when attributing multiple samples x. We also use gradient compression; we found that compression by quantisation works much better for diffusion models compared to the SVD-based compression used by[26] (see Appendix C), likely due to the fact that gradients $\nabla_\theta \ell(\theta, x_n)$ are not low-rank in this setting.

3.3. What to measure

For diffusion models, arguably the most natural question to ask might be, for a given sample x generated from the model, how did the training samples influence the probability of generating a sample x? For example, in the context of copyright infringement, we might want to ask if removing certain copyrighted works would substantially reduce the probability of generating x. With influence functions, these questions could be interpreted as setting the measurement function $m(\theta, x)$ to be the (marginal) log-probability of generating x from the diffusion model: $\log p_\theta(x)$.

Computing the marginal log-probability introduces some challenges. Diffusion models have originally been designed with the goal of tractable sampling, and not log-likelihood evaluation.[13][17] only introduce a lower-bound on the marginal log-probability.[20] show that exact log-likelihood evaluation is possible, but it only makes sense in settings where the training data distribution has a density (e.g. uniformly dequantised data), and it only corresponds to the marginal log-likelihood of the model when sampling deterministically[21].[7] Also, taking gradients of that measurement, as required for influence functions, is non-trivial. Hence, in most cases, we might need a proxy measurement for the marginal probability. We consider a couple of proxies in this work:

1. **Loss.** Approximate $\log p_\theta(x)$ with the diffusion loss $\ell(\theta, x)$ in Equation (2) on that particular example. This corresponds to the ELBO with reweighted per-timestep loss terms (see Figure 18).

2. **Probability of sampling trajectory.** If the entire sampling trajectory $x^{(0:T)}$ that generated sample x is available, consider the probability of that trajectory $p_\theta(x^{(0:T)}) = p(x^T) \prod_{t=1}^{T} p_\theta(x^{(t-1)}|x^{(t)})$.
3. **ELBO.** Approximate $\log p_\theta(x)$ with an Evidence Lower-Bound (eq. (5))[13].

Initially, we might expect ELBO to be the best motivated proxy, as it is the only one with a clear link to the marginal log-probability. Probability of sampling trajectory might also appear sensible, but it doesn't take into account the fact that there are multiple trajectories $x^{(0:T)}$ that all lead to the same final sample $x^{(0)}$, and it has the disadvantage of not being reparameterisation invariant.[8] We empirically investigate these different proxies in Section 4.

4. Experiments

Evaluating Data Attribution.

To evaluate the proposed data attribution methods, we primarily focus on two metrics: *Linear Data Modelling Score* (LDS) and *retraining without top influences*. LDS measures how well a given attribution method can predict the relative magnitude in the change in a measurement as the model is retrained on (random) subsets of the training data. For an attribution method $a(\mathcal{D}, \mathcal{D}', x)$ that approximates how a measurement $m(\theta^*(\mathcal{D}), x)$ would change if a model was trained on an altered dataset $\mathcal{D}'$, LDS measures the Spearman rank correlation between the predicted change in output and actual change in output after retraining on different subsampled datasets:

$$\text{spearman}\left[\left(a(\mathcal{D}, \tilde{\mathcal{D}}_i, x)\right)_{i=1}^{M}; \left(m(\theta^*(\tilde{\mathcal{D}}_i), x)\right)_{i=1}^{M}\right],$$

where $\tilde{\mathcal{D}}_i$ are independently subsampled versions of the original dataset $\mathcal{D}$, each containing 50% of the points sampled without replacement. However, a reality of deep learning is that, depending on the random seed used for initialisation and setting the order in which the data is presented in training, training on a fixed dataset can produce different models with functionally different behaviour. Hence, for any given dataset $\mathcal{D}'$, different measurements could be obtained depending on the random seed used. To mitigate the issue, [16] propose to use an ensemble average measurement after retraining as the "oracle" target:

$$\text{LDS} = \text{spearman}\left[\left(a(\mathcal{D}, \tilde{\mathcal{D}}_i, x)\right)_{i=1}^{M}; \left(\frac{1}{K}\sum_{k=1}^{K} m(\tilde{\theta}_k^*(\tilde{\mathcal{D}}_i), x)\right)_{i=1}^{M}\right], \tag{13}$$

where $\tilde{\theta}_k^*(\mathcal{D}') \in \mathbb{R}^{d_{\text{param}}}$ are the parameters resulting from training on $\mathcal{D}'$ with a particular seed k.

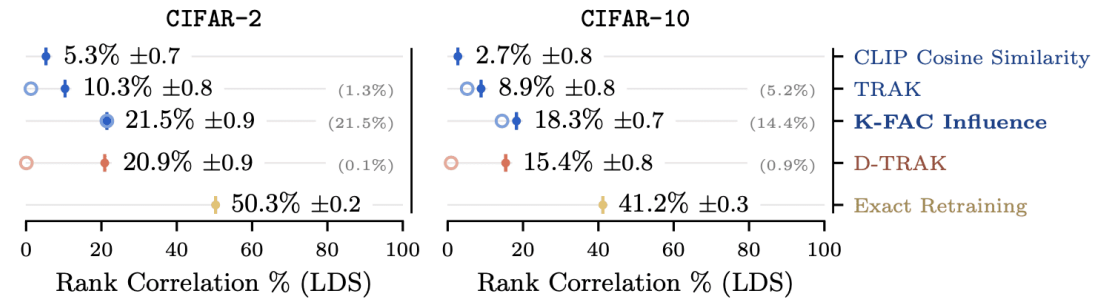
Retraining without top influences, on the other hand, evaluates the ability of the data attribution method to surface the most influential data points – namely, those that would most negatively affect the measurement $m(\theta^*(\mathcal{D}'), x)$ under retraining from scratch on a dataset $\mathcal{D}'$ with these data points removed. For each method, we remove a fixed percentage of the most influential datapoints from $\mathcal{D}$ to create the new dataset $\mathcal{D}'$, and report the change in the measurement $m(\theta^*(\mathcal{D}'), x)$ relative to $m(\theta^*(\mathcal{D}), x)$ (measurement by the model trained on the full dataset $\mathcal{D}$).

In all experiments, we look at measurements on samples generated by the model trained on $\mathcal{D}$. We primarily focus on Denoising Diffusion Probabilistic Models (DDPM) [13] throughout.[8]

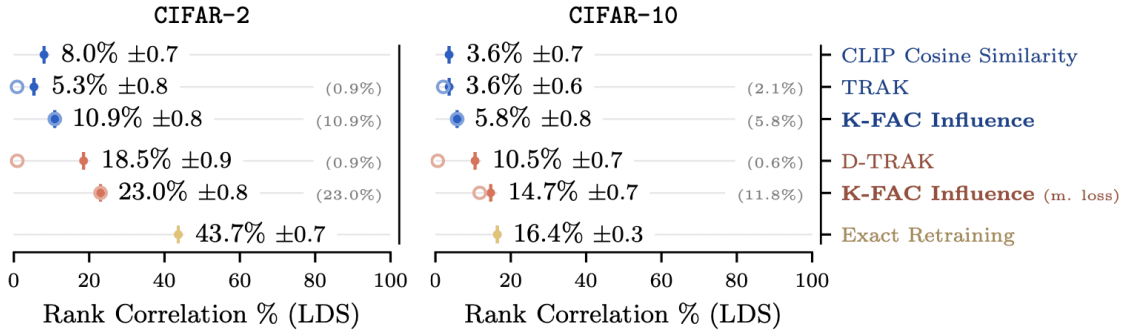
Baselines

We compare influence functions with K-FAC and $\text{GGN}_{\mathcal{D}}^{\text{model}}$ (MC-Fisher; 10) as the Hessian approximation (K-FAC Influence) to TRAK as formulated for diffusion models in [14][15]. In our framework, their method can be tersely described as using $\text{GGN}_{\mathcal{D}}^{\text{loss}}$ (Empirical Fisher) in Equation (11) as a Hessian approximation instead of

$\mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}$ (MC-Fisher) in Equation (10), and computing the Hessian-preconditioned inner products using random projections [28] rather than K-FAC. We also compare to the ad-hoc changes to the measurement/training loss in the influence function approximation (D-TRAK) that were shown by [15] to give improved performance on LDS benchmarks. Note that, the changes in D-TRAK were directly optimised for improvements in LDS scores in the diffusion modelling setting, and lack any theoretical motivation. Hence, a direct comparison for the changes proposed in this work (K-FAC Influence) is TRAK; the insights from D-TRAK are orthogonal to our work. These are the only prior works motivated by predicting the change in a model's measurements after retraining that have been applied to the general diffusion modelling setting that we are aware of. We also compare to naïvely using cosine similarity between the CLIP [29] embeddings of the training datapoints and the generated sample as a proxy for influence on the generated samples. Lastly, we report LDS results for the oracle method of "Exact Retraining", where we actually retraining a single model to predict the changes in measurements.



(a) LDS results on the **loss** measurement.



(b) LDS results on the **ELBO** measurement.

Figure 2. Linear Data-modelling Score (LDS) for different data attribution methods. Methods that substitute in *incorrect* measurement functions into the approximation are separated and plotted with [red circle]. We plot results for both the best Hessian-approximation damping value with [gray circle] and a "default" damping value with [gray empty circle] (where applicable). Where applicable, the numerical results are reported in black for the best damping value, and for the "default" damping value in (gray). "(m. loss)" implies that the appropriate measurement function was substituted with the loss $\ell(\theta, x)$ measurement function in the approximation. Results for the exact retraining method (oracle), are shown with [gold circle] ;. Standard error in the LDS score estimate is indicated with '±', where the mean is taken over different generated samples x on which the change in measurement is being estimated.

LDS

The LDS results attributing the loss and ELBO measurements are shown in Figures 2a and 2b. K-FAC Influence outperforms TRAK in all settings. K-FAC Influence using the loss measurement also outperforms the benchmark-tuned changes in D-TRAK in all settings as well. In Figures 2a and 2b, we report the results for both the best damping values from a sweep (see Appendix D), as well as for "default" values following recommendations in previous work (see Appendix G.4). TRAK and D-TRAK appear to be significantly more sensitive to tuning the damping factor than K-FAC Influence. They often don't perform at all if the damping

doi.org/10.32388/BOJDXM

factor is too small, and take a noticeable performance hit if the damping factor is not tuned to the problem or method (see Figures 8 and 10 in Appendix D). However, in most applications, tuning the damping factor would be infeasible, as it requires retraining the model many times over to construct an LDS benchmark, so this is a significant limitation. In contrast, for K-FAC Influence, we find that generally any sufficiently small value works reasonably well if enough samples are taken for estimating the loss and measurement gradients (see Figures 7 and 9).

Retraining without top influences

The counterfactual retraining results are shown in Figure 3 for CIFAR-2, CIFAR-10, with 2% and 10% of the data removed. In this evaluation, influence functions with K-FAC consistently pick more influential training examples (i.e. those which lead to a higher loss reduction) than the baselines.

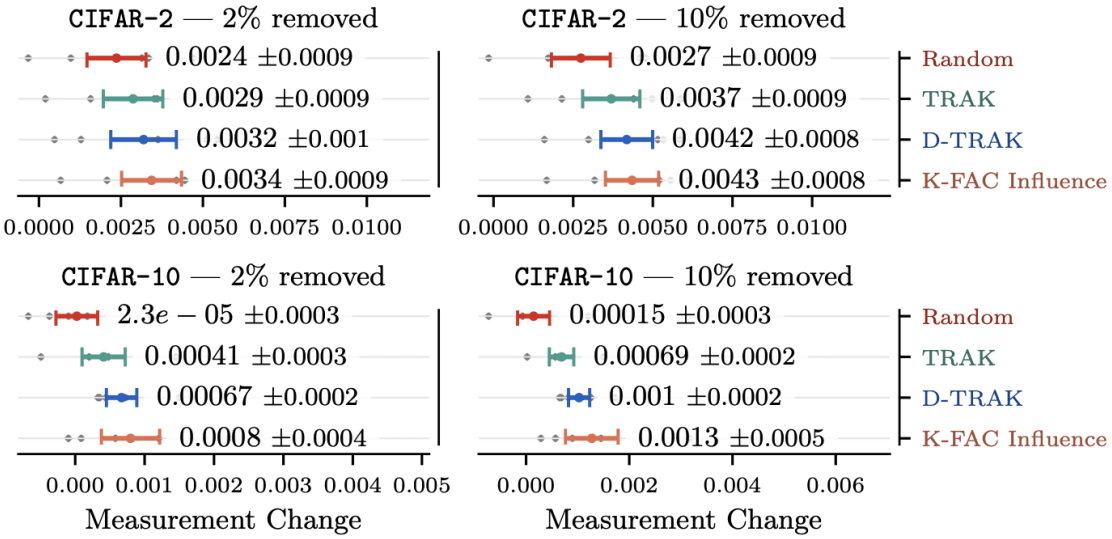


Figure 3. Changes in measurements under counterfactual retraining without top influences for the **loss** measurement. The standard error in the estimate of the mean is indicated with error bars and reported after '±', where the average is over different generated samples for which top influences are being identified.

Hessian Approximation Ablation

In Figure 4, we explore the impact of the Hessian approximation design choices discussed in Section 3.1. We use K-FAC to approximate the GGN in all cases, with either the "expand" or the "reduce" variant (Section 3.1.2). We find that the better-motivated "MC-Fisher" estimator $\mathrm{GGN}^{\mathrm{model}}$ in Equation (9) does indeed perform better than the "empirical Fisher" in Equation (11) used in TRAK and D-TRAK. Secondly, we find that K-FAC expand significantly outperforms K-FAC reduce, which stands in contrast to the results in the second-order

optimisation setting where the two are on par with one another[12]. There are multiple differences from our setting to the one from the previous optimisation results: we use a square loss instead of a cross entropy loss, a full dataset estimate, a different architecture, and evaluate the approximation in a different application. Notably, the expand variant is the better justified one since the diffusion modelling problem corresponds to the expand setting in[12]. Hence, our results all seem to imply that a better Hessian approximation directly results in better downstream data attribution performance. However, we do not directly evaluate the approximation quality of the estimates and also do not sweep over the damping value for all variants.

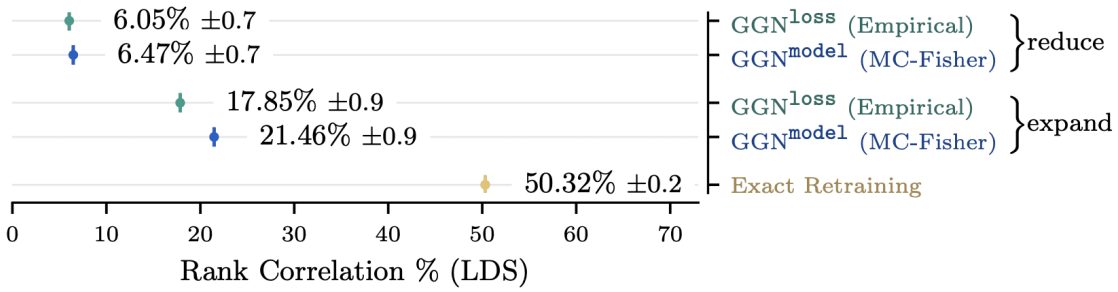


Figure 4. Ablation over the different Hessian approximation variants introduced in Section 3.1. We ablate two versions of the GGN: the "MC" Fisher in Equation (9) and the "Empirical" Fisher in Equation (11), as well as two settings for the K-FAC approximation: "expand" and "reduce"[12].

4.1. Potential challenges to use of influence functions for diffusion models

One peculiarity in the LDS results, similar to the findings in[15], is that substituting the loss measurement for the ELBO measurement when predicting changes in ELBO actually works better than using the correct measurement (see Figure 2b "K-FAC Influence (measurement loss)").[9] To try and better understand the properties of influence functions for predicting the numerical change in behaviour after retraining, in this section we perform multiple ablations and report different interesting phenomena resulting from these that give some insight into the challenges of using influence functions in this setting.

As illustrated in Figure 18, gradients of the ELBO and training loss measurements, up to a constant scaling, consist of the same per-diffusion-timestep loss term gradients $\nabla_\theta \ell_t(\theta, x)$, but with a different weighting. To try and break-down why approximating the change in ELBO with the training loss measurement gives higher LDS scores, we first look at predicting the change in the per-diffusion-timestep losses ℓ_t while substituting *different* per-diffusion-timestep losses into the K-FAC influence approximation. The results are shown in Figure 11, leading to the following observation:

Observation 1. *Higher-timestep losses $\ell_t(\theta, x)$ act as better proxies for lower-timestep losses.*

More specifically, changes in losses ℓ_t can in general be well approximated by substituting measurements $\ell_{t'}$ into the influence approximation with $t' > t$. In some cases, using the *incorrect* timestep $t' > t$ even results in significantly better LDS scores than the correct timestep $t' = t$.

Based on Observation 1, it is clear that influence function-based approximations have limitations when being applied to predict the numerical change in loss measurements. We observe another pattern in how they can fail:

Observation 2. *Influence functions predict both positive and negative influence on loss, but, in practice, removing data points predominantly increases loss.*

We show in Figures 15 and 16 that influence functions tend to overestimate how often removal of a group data points will lead to improvements in loss on a generated sample (both for aggregate diffusion training loss in Section 2.1, and the per-diffusion-timestep loss in Equation (2)).

Lastly, although ELBO is perhaps the measurement with the most direct link to the marginal probability of sampling a particular example, we find it has some peculiar properties. The below observation particularly puts its usefulness as a measurement function into question:

Observation 3.*ELBO is close to constant on generated samples, irrespective of which examples were removed from the training data.*

As illustrated in Figure 17, ELBO measurement is close to constant for any given sample generated from the model, no matter which 50% subset of the training data is removed. In particular, it is extremely rare that if one sample is more likely to be generated than another sample by one model (as measured by ELBO), it will be less likely to be generated than another by a model trained on a different random subset of the data. This implies that the standard DDPM ELBO might be a poor proxy for the real question we intend to ask: "how likely is a given sample to be generated by a model?". It would appear improbable that the true probabilities of generating any given sample are close-to unaffected by resampling what data subset the model was trained on.

5. Discussion

In this work, we extended the influence functions approach to the diffusion modelling setting, and showed different ways in which the GGN Hessian approximation can be formulated in this setting. Our proposed method with recommended design choices improves performance copmared to existing techniques across various data attribution evaluation metrics. Nonetheless, experimentally, we are met with two contrasting findings: on the one hand, influence functions in the diffusion modelling setting appear to be able to identify important influences. The surfaced influential examples do significantly impact the training loss when retraining the model without them (Figure 3), and they appear perceptually very relevant to the generated samples. On the other hand, they fall short of accurately predicting the numerical changes in measurements after retraining. Thes appears to be especially the case for measurement functions we would argue are most relevant in the image generative modelling setting – proxies for marginal probability of sampling a particular example. This appears to be both due to the limitations of the influence functions approximation, but also due to the shortcomings of the considered proxy measurements (Section 4.1).

Despite these shortcomings, influence functions can still offer valuable insights: they can serve as a useful exploratory tool for understanding model behaviour in a diffusion modelling context, and can help guide data curation, identifying examples most responsible for certain behaviours. To make them useful in settings where numerical accuracy in the predicted behaviour after retraining is required, such as copyright infringement, we believe more work is required into 1)finding better proxies for marginal probability than ELBO and probability of sampling trajectory , and 2) even further improving the influence function approximation.

Appendix A. Derivation of Influence Functions

In this section, we state the implicit function theorem (Appendix A.1). Then, in Appendix A.2, we introduce the details of how it can be applied in the context of a loss function $\mathcal{L}(\epsilon, \theta)$ parameterised by a continuous hyperparameter ϵ (which is, e.g., controlling how down-weighted the loss terms on some examples are, as in Section 2.2).

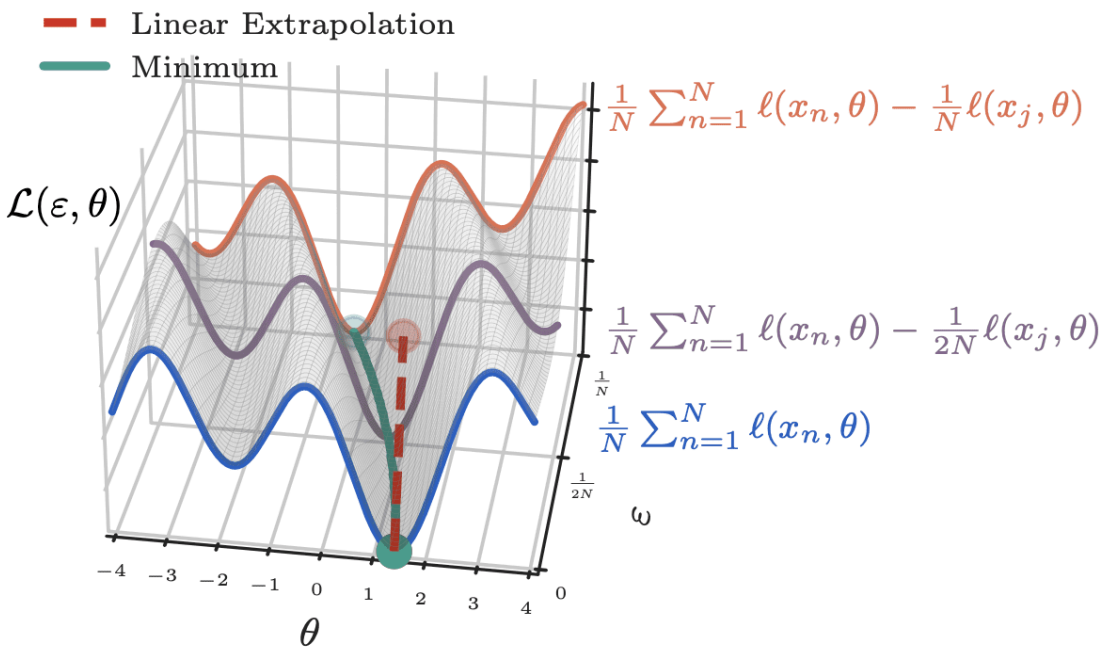


Figure 5. Illustration of the influence function approximation for a 1-dimensional parameter space $\theta \in \mathbb{R}$. Influence funcitons consider the extended loss landscape $\mathcal{L}(\epsilon, \theta) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{n=1}^{N}\ell(x_n,\theta) - \epsilon\ell(x_j,\theta)$, where the loss $\ell(x_j,\theta)$ for some datapoint x_j (alternatively, group of datapoints) is down-weighted by ϵ. By linearly extrapolating how the optimal set of parameters θ would change around $\epsilon = 0$ (ω), we can predicted how the optimal parameters would change when the term $\ell(x_j,\theta)$ is fully removed from the loss (ω_0).

A.1. Implicit Function Theorem

Theorem 1 (Implicit Function Theorem[23]) *Let $F : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$ be a continuously differentiable function, and let $\mathbb{R}^n \times \mathbb{R}^m$ have coordinates $(\mathbf{x}, \mathbf{y})$. Fix a point $(\mathbf{a}, \mathbf{b}) = (a_1, \ldots, a_n, b_1, \ldots, b_m)$ with $F(\mathbf{a}, \mathbf{b}) = \mathbf{0}$, where $\mathbf{0} \in \mathbb{R}^m$ is the zero vector. If the Jacobian matrix $\nabla_{\mathbf{y}} F(\mathbf{a}, \mathbf{b}) \in \mathbb{R}^{m\times m}$ of $\mathbf{y} \mapsto F(\mathbf{a}, \mathbf{y})$*

$$[\nabla_{\mathbf{y}} F(\mathbf{a}, \mathbf{b})]_{ij} = \frac{\partial F_i}{\partial y_j}(\mathbf{a}, \mathbf{b}),$$

is invertible, then there exists an open set $U \subset \mathbb{R}^n$ containing $\mathbf{a}$ such that there exists a unique function $g : U \to \mathbb{R}^m$ such that $g(\mathbf{a}) = \mathbf{b}$, and $F(\mathbf{x}, g(\mathbf{x})) = \mathbf{0}$ for all $\mathbf{x} \in U$. Moreover, g is continuously differentiable.

Remark 1 (Derivative of the implicit function) *Denoting the Jacobian matrix of $\mathbf{x} \mapsto F(\mathbf{x}, \mathbf{y})$ as:*

$$[\nabla_{\mathbf{x}} F(\mathbf{x}, \mathbf{y})]_{ij} = \frac{\partial F_i}{\partial x_j}(\mathbf{x}, \mathbf{y}),$$

the derivative $\frac{\partial \mathbf{g}}{\partial \mathbf{x}} : U \to \mathbb{R}^{m\times n}$ of $g : U \to \mathbb{R}^m$ in Theorem 1 can be written as:

$$\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} = -[\nabla_{\mathbf{y}} F(\mathbf{x}, g(\mathbf{x}))]^{-1} \nabla_{\mathbf{x}} F(\mathbf{x}, g(\mathbf{x})). \tag{14}$$

doi.org/10.32388/BOJDXM

This can readily be seen by noting that, for $\mathbf{x} \in U$:

$$F(\mathbf{x}', g(\mathbf{x}')) = \mathbf{0} \quad \forall \mathbf{x}' \in U \Rightarrow \frac{dF(\mathbf{x}, g(\mathbf{x}))}{d\mathbf{x}} = \mathbf{0}.$$

Hence, since g is differentiable, we can apply the chain rule of differentiation to get:

$$\mathbf{0} = \frac{dF(\mathbf{x}, g(\mathbf{x}))}{d\mathbf{x}} = \nabla_{\mathbf{x}} F(\mathbf{x}, g(\mathbf{x})) + \nabla_{\mathbf{y}} F(\mathbf{x}, g(\mathbf{x})) \frac{\partial g(\mathbf{x})}{\partial \mathbf{x}}.$$

Rearranging gives equation (14).

A.2. Applying the implicit function theorem to quantify the change in the optimum of a loss

Consider a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ that depends on some hyperparameter $\epsilon \in \mathbb{R}^n$ (in Section 2.2, this was the scalar by which certain loss terms were down-weighted) and some parameters $\theta \in \mathbb{R}^m$. At the minimum of the loss function $\mathcal{L}(\epsilon, \theta)$, the derivative with respect to the parameters θ will be zero. Hence, assuming that the loss function is twice continuously differentiable (hence $\frac{\partial L}{\partial \epsilon}$ is continuously differentiable), and assuming that for some $\epsilon' \in \mathbb{R}^n$ we have a set of parameters $\theta^\star$ such that $\frac{\partial \mathcal{L}}{\partial \epsilon}(\epsilon', \theta^\star) = \mathbf{0}$ and the Hessian $\frac{\partial^2 \mathcal{L}}{\partial \theta^2}(\epsilon', \theta^\star)$ is invertible, we can apply the implicit function theorem to the derivative of the loss function $\frac{\partial \mathcal{L}}{\partial \epsilon} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, to get the existence of a continuously differentiable function g such that $\frac{\partial \mathcal{L}}{\partial \epsilon}(\epsilon, g(\epsilon)) = \mathbf{0}$ for ϵ in some neighbourhood of ϵ'.

Now $g(\epsilon)$ might not necessarily be a minimum of $\theta \mapsto \mathcal{L}(\epsilon, \theta)$. However, by making the further assumption that $\mathcal{L}$ is strictly convex we can ensure that whenever $\frac{\partial \mathcal{L}}{\partial \theta}(\epsilon, \theta) = \mathbf{0}$, θ is a unique minimum, and so $g(\epsilon)$ represents the change in the minimum as we vary ε. This is summarised in the lemma below:

Lemma 1. *Let $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be a twice continuously differentiable function, with coordinates denoted by $(\epsilon, \theta) \in \mathbb{R}^n \times \mathbb{R}^m$, such that $\theta \mapsto \mathcal{L}(\epsilon, \theta)$ is strictly convex $\forall \epsilon \in \mathbb{R}^n$. Fix a point $(\epsilon', \theta^\star)$ such that $\frac{\partial \mathcal{L}}{\partial \theta}(\epsilon', \theta^\star) = \mathbf{0}$. Then, by the Implicit Function Theorem applied to $\frac{\partial \mathcal{L}}{\partial \theta}$, there exists an open set $U \subset \mathbb{R}^n$ containing $\theta^\star$ such that there exists a unique function $g : U \to \mathbb{R}^m$ such that $g(\epsilon') = \theta^\star$, and $g(\epsilon)$ is the unique minimum of $\theta \mapsto \mathcal{L}(\epsilon, \theta)$ for all $\epsilon \in U$. Moreover, g is continuously differentiable with derivative:*

$$\frac{\partial g(\varepsilon)}{\partial \varepsilon} = -\left[\frac{\partial^2 \mathcal{L}}{\partial \theta^2}(\varepsilon, g(\varepsilon))\right]^{-1} \frac{\partial^2 \mathcal{L}}{\partial \varepsilon \partial \theta}(\varepsilon, g(\varepsilon)) \tag{15}$$

Remark 2. *For a loss function $\mathcal{L} : \mathbb{R} \times \mathbb{R}^m$ of the form $\mathcal{L}(\varepsilon, \theta) = \mathcal{L}_1(\theta) + \varepsilon \mathcal{L}_2(\theta)$ (such as that in (4)), $\frac{\partial^2 \mathcal{L}}{\partial \varepsilon \partial \theta}(\varepsilon, g(\varepsilon))$ in the equation above simplifies to:*

$$\frac{\partial^2 \mathcal{L}}{\partial \varepsilon \partial \theta}(\varepsilon, g(\varepsilon)) = \frac{\partial \mathcal{L}_2}{\partial \theta}(g(\varepsilon)) \tag{16}$$

The above lemma and remark give the result in Equation (5). Namely, in section 2.2:

$$\mathcal{L}(\varepsilon,\theta)=\underbrace{\frac{1}{N}\sum_{i=1}^{N}\ell(\theta,x_i)}_{\mathcal{L}_1}-\overbrace{\frac{1}{M}\sum_{j=1}^{M}\ell(\theta,x_{i_j})\varepsilon}^{\mathcal{L}_2}\qquad\overset{\text{eq. (16)}}{\Longrightarrow}\qquad\frac{\partial^2\mathcal{L}}{\partial\varepsilon\partial\theta}=-\frac{1}{M}\sum_{j=1}^{M}\frac{\partial}{\partial\theta}\ell(\theta,x_{i_j})$$

$$\overset{\text{eq. (15)}}{\Longrightarrow}\qquad\frac{\partial g(\varepsilon)}{\partial\varepsilon}=\left[\frac{\partial^2\mathcal{L}}{\partial\theta^2}(\varepsilon,g(\varepsilon))\right]^{-1}\frac{1}{M}\sum_{j=1}^{M}\frac{\partial}{\partial\theta}\ell(\theta,x_{i_j})$$

Appendix B. Derivation of the Fisher "GGN" formulation for Diffusion Models

As discussed in Section 2.2.1 partitioning the function $\theta\mapsto\|\epsilon^{(t)}-\epsilon_\theta^t(x^{(t)})\|^2$ into the model output $\theta\mapsto\epsilon_\theta^t(x^{(t)})$ and the ℓ_2 loss function is a natural choice and results in

$$\begin{aligned}\mathrm{GGN}_{\mathcal{D}}^{\text{model}}(\theta)&=\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})}}\left[\nabla_\theta^\top\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon^{(\tilde{t})}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\nabla_\theta\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right]\right]\\&=\frac{2}{N}\sum_{n=1}^{N}\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})}}\left[\nabla_\theta^\top\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\nabla_\theta\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right]\right].\end{aligned}\tag{17}$$

Note that we used

$$\frac{1}{2}\nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon^{(\tilde{t})}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2=I.$$

We can substitute I with

$$I=\mathbb{E}_{\epsilon_{\text{mod}}}\left[-\frac{1}{2}\nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\log p(\epsilon_{\text{mod}}|\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})}))\right],\qquad p(\epsilon_{\text{mod}}|\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})}))=\mathcal{N}(\epsilon_{\text{mod}}|\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})}),I),$$

where the mean of the Gaussian is chosen to be the model output $\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})$. Furthermore, by using the "score" trick:

$$\begin{aligned}\mathbb{E}_{\epsilon_{\text{mod}}}\left[\nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\log p\left(\epsilon_{\text{mod}}\mid\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right)\right]&=-\mathbb{E}_{\epsilon_{\text{mod}}}\left[\nabla_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\log p\left(\epsilon_{\text{mod}}\mid\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right)\nabla^\top_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\log p\left(\epsilon_{\text{mod}}\mid\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right)\right]\\&=-\mathbb{E}_{\epsilon_{\text{mod}}}\left[\frac{1}{2}\nabla_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\frac{1}{2}\nabla^\top_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\right],\end{aligned}$$

we can rewrite:

$$\begin{aligned}\nabla_\theta^\top\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\nabla_\theta\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})&=-2\nabla_\theta^\top\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\mathbb{E}_{\epsilon_{\text{mod}}}\left[\nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\log p\left(\epsilon_{\text{mod}}\mid\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right)\right]\nabla_\theta\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\\&=\frac{1}{2}\mathbb{E}_{\epsilon_{\text{mod}}}\left[\nabla_\theta^\top\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\nabla_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\frac{1}{2}\nabla^\top_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})}\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\nabla_\theta\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right]\\&=\frac{1}{2}\mathbb{E}_{\epsilon_{\text{mod}}}\left[\nabla_\theta\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\nabla_\theta^\top\|\epsilon_{\text{mod}}-\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2\right],\end{aligned}$$

where the last equality follows by the chain rule of differentiation. We can thus rewrite the expression for the GGN in (17) as

doi.org/10.32388/BOJDXM

$$\mathrm{GGN}_{\mathcal{D}}^{\text{model}}(\theta) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})},\epsilon_{\text{mod}}}\left[\nabla_\theta g_n(\theta)\nabla_\theta g_n(\theta)^\top\right]\right]$$
$$g(\theta) \stackrel{\text{def}}{=} \|\epsilon_{\text{mod}} - \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\|^2.$$

Appendix C. Gradient compression ablation

In Figure 6, we ablate different compression methods by computing the per training datapoint influence scores with compressed query (measurement) gradients, and looking at the Pearson correlation and the rank correlation to the scores compute with the uncompressed gradients. We hope to see a correlation of close to 100%, in which case the results for our method would be unaffected by compression. We find that using quantisation for compression results in almost no change to the ordering over training datapoints, even when quantising down to 8 bits. This is in contrast to the SVD compression scheme used in [26]. This is likely because the per-example gradients naturally have a low-rank (Kronecker) structure in the classification, regression, or autoregressive language modelling settings, such as that in [26]. On the other hand, the diffusion training loss and other measurement functions considered in this work do not have this low-rank structure. This is because computing them requires multiple forward passes; for example, for the diffusion training loss we need to average the mean-squared error loss in Equation (2) over multiple noise samples $\epsilon^{(t)}$ and multiple diffusion timesteps. We use 8 bit quantisation with query gradient batching [26] for all KFAC experiments throughout this work.

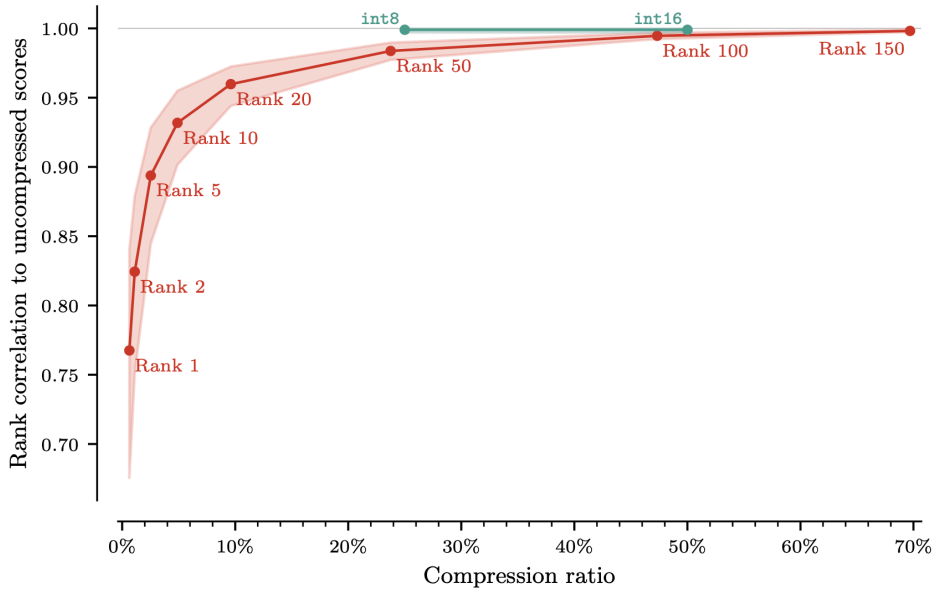


Figure 6. Comparison of gradient compression methods for the influence function approximation.

doi.org/10.32388/BOJDXM

Appendix D. Damping LDS ablations

We report an ablation over the LDS scores with GGN approximated with different damping factors for TRAK/D-TRAK and K-FAC influence in Figures 7 to 10. The reported damping factors for TRAK are normalised by the dataset size so that they correspond to the equivalent damping factors for our method when viewing TRAK as an altenrative approximation to the GGN (see Section 3.1).

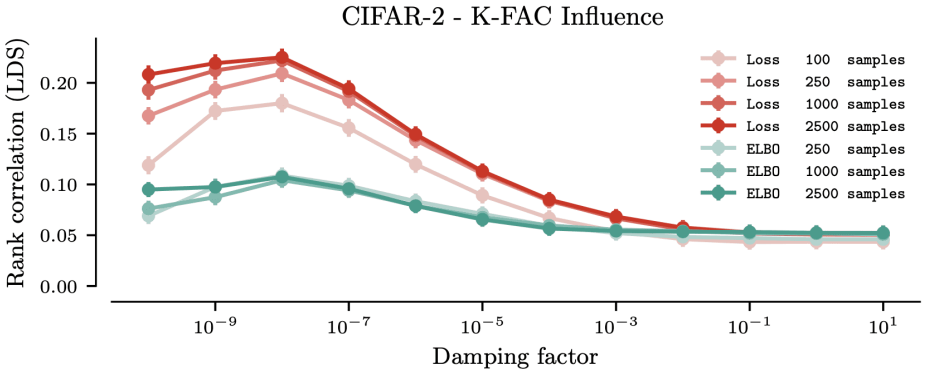


Figure 7. Effect of damping on the LDS scores for **K-FAC influence** on **CIFAR-2**. In this plot, K-FAC GGN approximation was always computed with 1000 samples, and the number of samples used for computing a Monte Carlo estimate of the training loss/measurement gradient is indicated on the legend.

doi.org/10.32388/BOJDXM

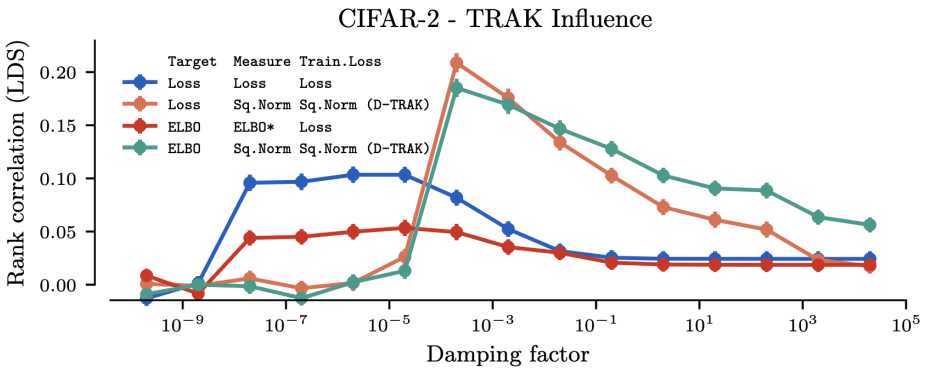


Figure 8. Effect of damping on the LDS scores for TRAK (random projection) based influence on `CIFAR-2`. 250 samples were used for Monte Carlo estiamtion of all quantities (GGN and the training loss/measurement gradients). In the legend: `Target` indicates what measurement we're trying to predict the change in after retraining, `Measure` indicates what measurement function was substituted into the influence function approximation, and `Train.Loss` indicates what function was substituted for the training loss in the computation of the GGN and gradient of the training loss in the influence function approximation.

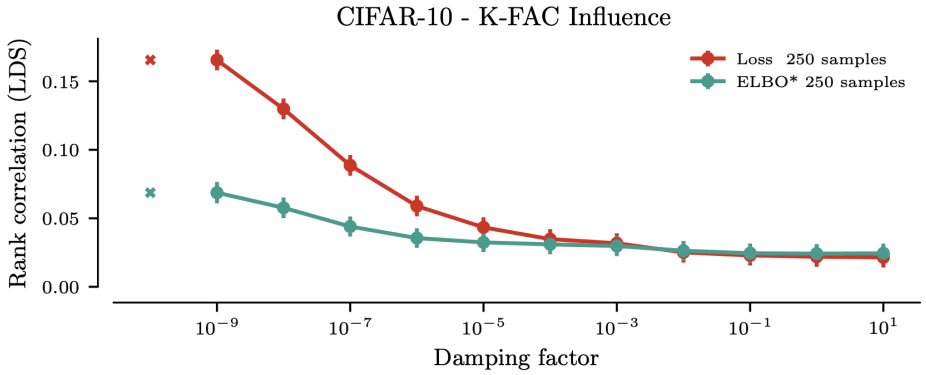


Figure 9. Effect of damping on the LDS scores for **K-FAC** based influence on **CIFAR-10**. 100 samples were used for computing the K-FAC GGN approximation, and 250 for computing a Monte Carlo estimate of the training loss/measurement gradients. × indicates a `NaN` result (the computation was not sufficiently numerically stable with that damping factor).

doi.org/10.32388/BOJDXM

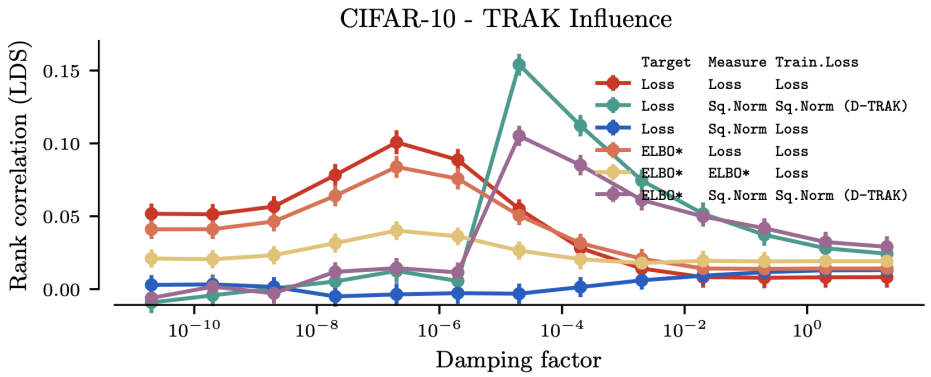


Figure 10. Effect of damping on the LDS scores for **TRAK** (random projection) based influence on **CIFAR-10**. 250 samples were used for Monte Carlo estiamtion of all quantities (GGN and the training loss/measurement gradients). In the legend: `Target` indicates what measurement we're trying to predict the change in after retraining, `Measure` indicates what measurement function was substituted into the influence function approximation, and `Train.Loss` indicates what function was substituted for the training loss in the computation of the GGN and gradient of the training loss in the influence function approximation.

Appendix E. Empirical ablations for challenges to use of influence functions for diffusion models

In this section, we describe the results for the observations discussed in Section 4.1.

Observation 1 is based on Figures 11 and 12. Figure 11 shows the LDS scores on CIFAR-2 when attributing per-timestep diffusion losses ℓ_t (see Equation (2)) using influence functions, whilst varying what (possibly wrong) per-timestep diffusion loss $\ell_{t'}$ is used as a measurement function in the influence function approximation (Equation (6)). Figure 12 is a counter-equivalent to Figure 16 where instead of using influence functions to approximate the change in measurement, we actually retrain a model on the randomly subsampled subset of data and compute the measurement.

doi.org/10.32388/BOJDXM

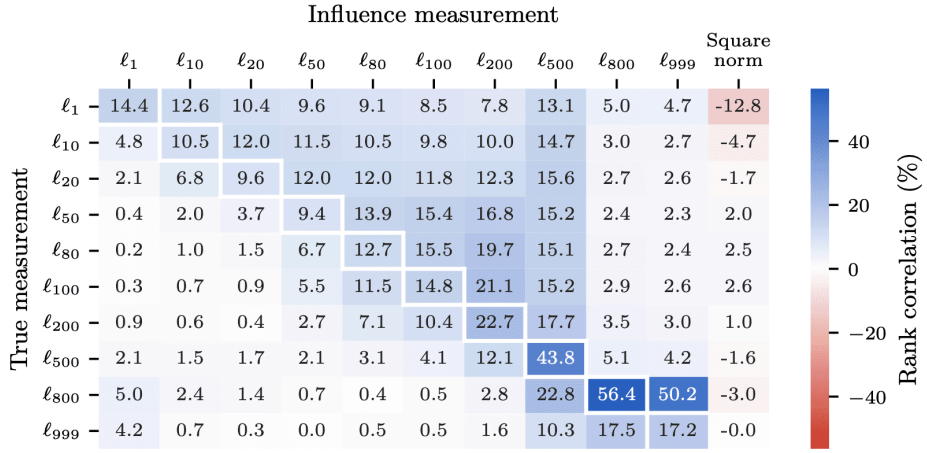


Figure 11. Rank correlation (LDS scores) between influence function estimates with different measurement functions and different true measurements CIFAR-2. The plot shows how well different per-timestep diffusion losses ℓ_t work as measurement functions in the influence function approximation, when trying to approximate changes in the actual measurements when retraining a model.

A natural question to ask with regards to Observation 1 is: does this effect go away in settings where the influence function approximation should more exact? Note that, bar the non-convexity of the training loss function $\mathcal{L}_{\mathcal{D}}$, the influence function approximation in Equation (6) is a linearisation of the actual change in the measurement for the optimum of the training loss functions with some examples down-weighted by ε around $\varepsilon = 0$. Hence, we might expect the approximation to be more exact when instead of fully removing some data points from the dataset (setting $\varepsilon = 1/N$), we instead down-weight their contribution to the training loss by a smaller non-zero factor. To investigate whether this is the case, we repeat the LDS analysis in Figures 11 and 12, but with $\varepsilon = 1/2N$; in other words, the training loss terms corresponding to the "removed" examples are simply down-weighted by a factor of $1/2$ in the retrained models. The results are shown in Figures 13 and 14. Perhaps somewhat surprisingly, a contrasting effect can be observed, where using per-timestep diffusion losses for larger times yields a higher absolute rank correlation, but with the opposing sign. The negative correlation between measurement $\ell_t, \ell_{t'}$ for $t \neq t'$ can also be observed for the true measurements in the retrained models in Figure 14. We also observe that in this setting, influence functions fail completely to predict changes in ℓ_t with the correct measurement function for $t \leq 200$.

doi.org/10.32388/BOJDXM

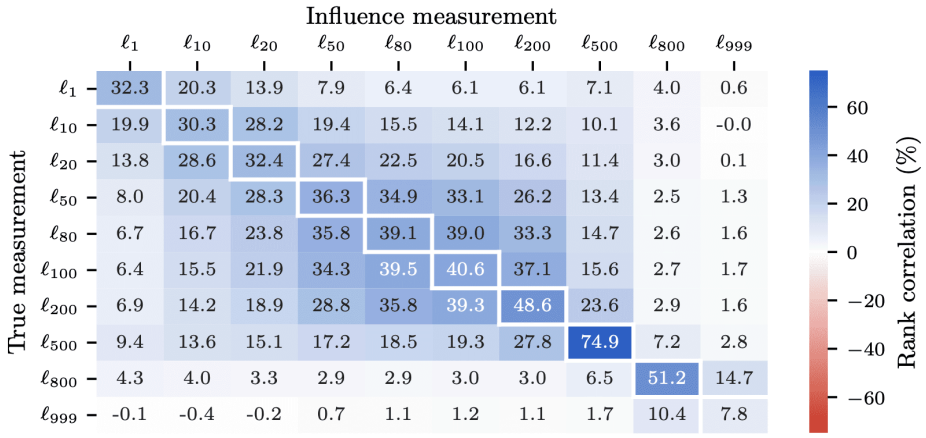


Figure 12. Rank correlation between true measurements for losses at different diffusion timesteps on CIFAR-2.

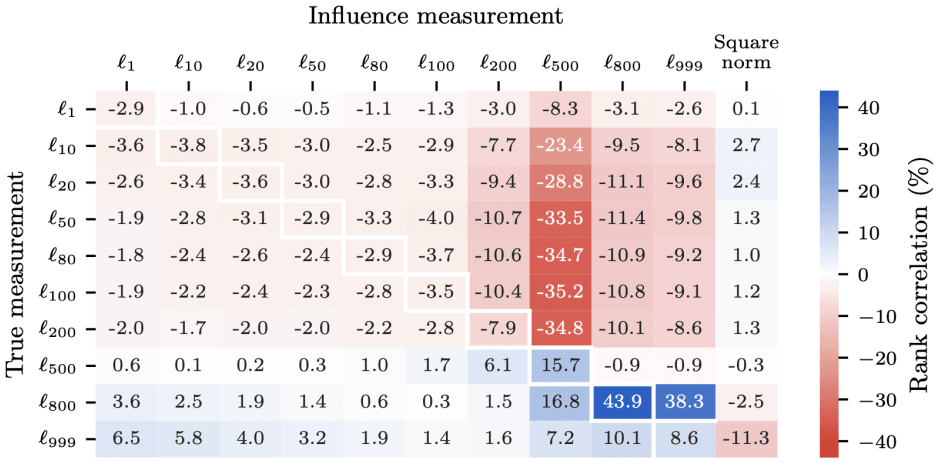


Figure 13. Rank correlation (LDS scores) between influence function estimates with different measurement functions and different true measurements CIFAR-2, but with the retrained models trained on the full dataset with a random subset of examples having a **down-weighted contribution to a training loss by a factor of** $\times 0.5$.

doi.org/10.32388/BOJDXM

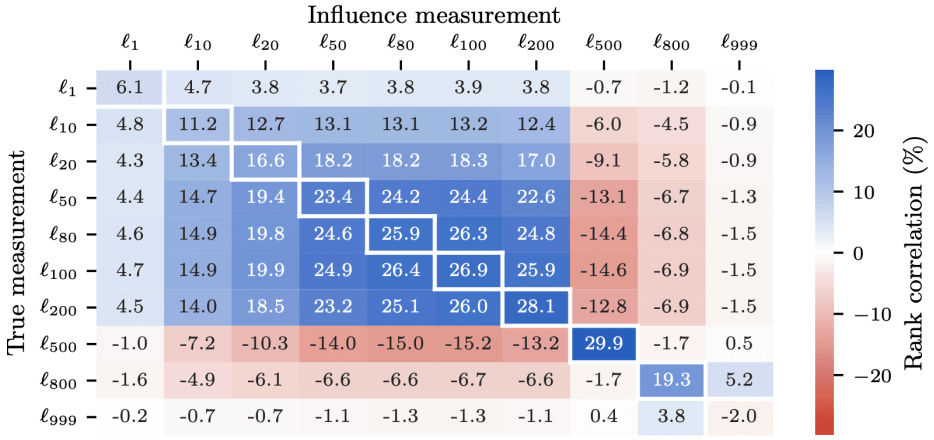


Figure 14. Rank correlation between true measurements for losses at different diffusion timesteps on CIFAR-2, but with the retrained models trained on the full dataset with a random subset of examples having a **down-weighted contribution to a training loss by a factor of** $\times 0.5$.

Observation 1. Figure 15 shows the changes in losses after retraining the model on half the data removed against the predicted changes in losses using K-FAC Influence for two datasets: CIFAR-2 and CIFAR-10. In both cases, for a vast majority of retrained models, the loss measurement on a sample increases after retraining. On the other hand, the influence functions predict roughly evenly that the loss will increase and decrease. This trend is amplified if we instead look at influence predicted for per-timestep diffusion losses ℓ_t (Equation (2)) for earlier timesteps t, which can be seen in Figure 16. On CIFAR-2, actual changes in $\ell_1, \ell_{50}, \ell_{100}$ measurements are actually always positive, which the influence functions approximation completely misses. For all plots, K-FAC Influence was ran with a damping factor of 10^{-8} and 250 samples for all gradient computations.

doi.org/10.32388/BOJDXM

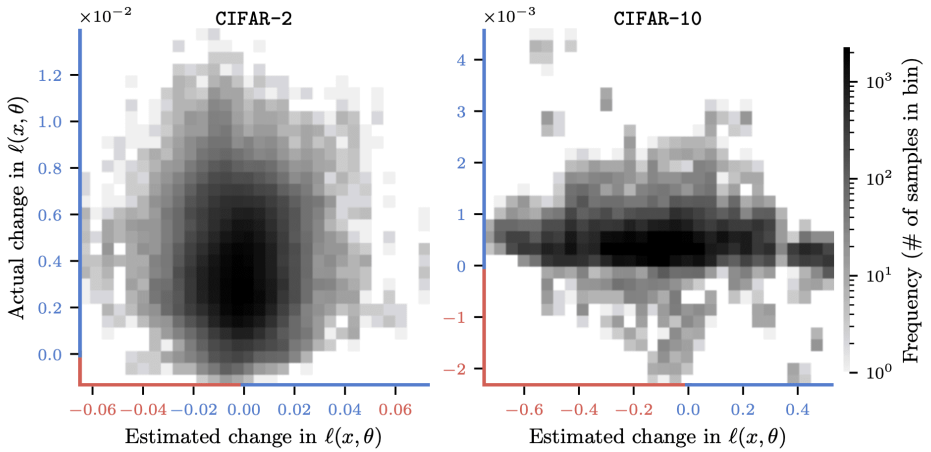


Figure 15. Change in diffusion loss ℓ in Section 2.1 when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for measurements $\ell(x, \theta)$ for 50 samples x generated from the diffusion model trained on all of the data. The scatter color indicates the sample x for which the change in measurement is plotted. The figure shows that influence functions tend to overestimate how often the loss will decrease when some training samples are removed; in reality, it happens quite rarely.

doi.org/10.32388/BOJDXM

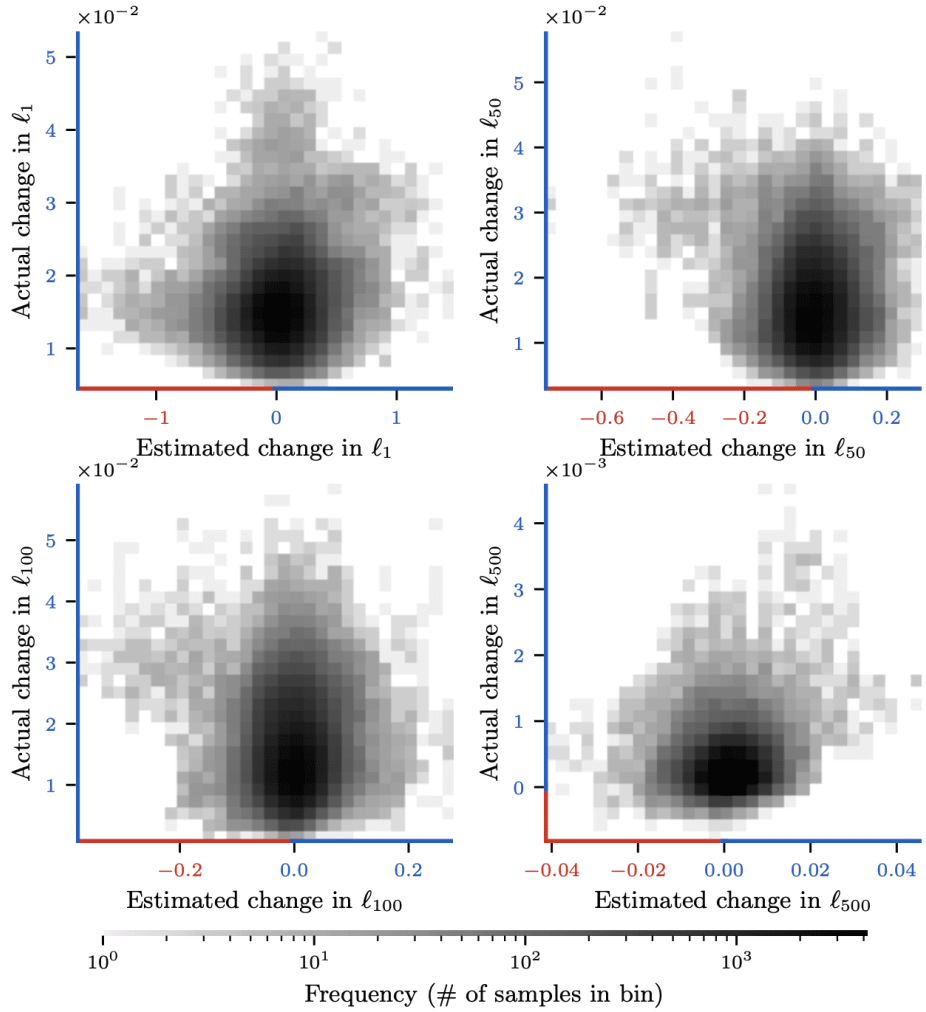


Figure 16. Change in per-diffusion-timestep losses ℓ_t when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for the CIFAR-2 dataset, for measurements $\ell_t(x, \theta)$ for 50 samples x generated from the diffusion model trained on all of the data. The scatter color indicates the sample x for which the change in measurement is plotted. The figure shows that: 1) influence functions predict that the losses ℓ_t will increase or decrease roughly equally frequently when some samples are removed, but, in reality, the losses almost always increase; 2) for sufficiently large time-steps (ℓ_{500}), this pattern seems to subside. Losses ℓ_t in the $200 - 500$ range seem to work well for predicting changes in other losses Figure 11.

Observation 3. Lastly, the observations that the ELBO measurements remain essentially constant for models trained on different subsets of data is based on Figure 17. There, we plot the values of the ELBO measurement

doi.org/10.32388/BOJDXM

for different pairs of models trained on different subsets of data, where we find near perfect correlation. The only pairs of models that exhibit an ELBO measurement correlation of less that 0.99 are the CIFAR-2 model trained on the full dataset compared to any model trained on a 50% subset, which is likely due to the fact that the 50% subset models are trained for half as many gradient iterations, and so may have not fully converged yet. For CIFAR-10, where we train for 5× as many training steps due to a larger dataset size, we observe near-perfect correlation in the ELBO measurements across all models. Each ELBO measurement was computed with a Monte-Carlo estimate using 5000 samples.

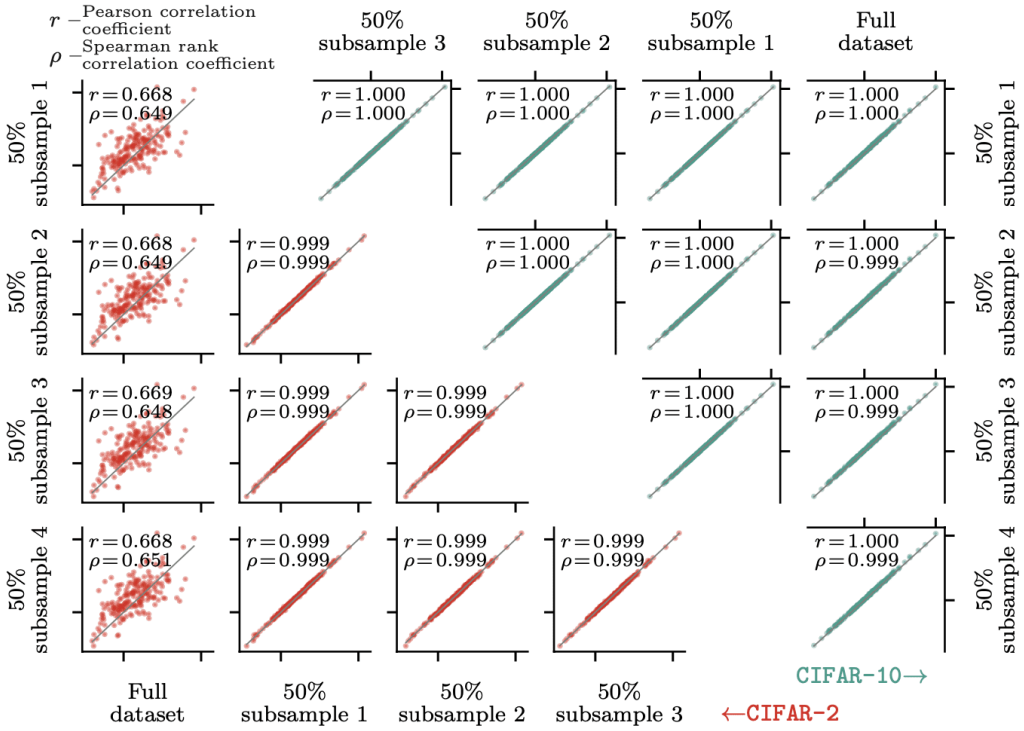


Figure 17. Correlation of the $\mathrm{ELBO}(x, \theta)$ measurements on different data points x (samples generated from the model trained on full data), for models trained on different subsets of data. Each subplot plots $\mathrm{ELBO}(x, \theta)$ measurements for 200 generated samples x, as measured by two models trained from scratch on different subsets of data, with the x-label and the y-label identifying the respective split of data used for training (either full dataset, or randomly subsampled 50%-subset). Each subplot shows the Pearson correlation coefficient (r) and the Spearman rank correlation (ρ) for the $\mathrm{ELBO}(x, \theta)$ measurements as measured by the two models trained on different subsets of data. The two parts of the figure show results for two different datasets: CIFAR-2 on the left, and CIFAR-10 on the right.

doi.org/10.32388/BOJDXM

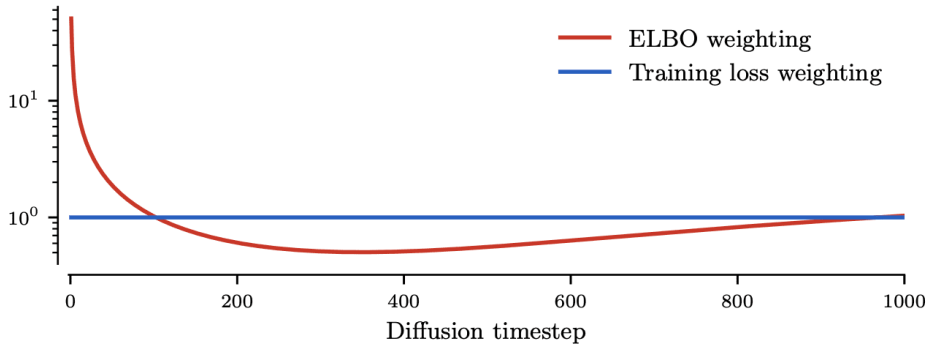


Figure 18. The diffusion loss and diffusion ELBO as formulated in [13] (ignoring the reconstruction term that accounts for the quantisation of images back to pixel space) are equal up to the weighting of the individual per-diffusion-timestep loss terms and a constant independent of the parameters. This plot illustrates the relatives difference in the weighting for per-diffusion-timestep losses applied in the ELBO vs. in the training loss.

Appendix F. LDS results for probability of sampling trajectory

The results for the "log probability of sampling trajectory" measurements are shown in Figure 19. The probability of sampling trajectory appears to be a measurement with a particularly low correlation across different models trained with the same data, but different random seeds. This is perhaps unsurprising, since the measurement comprises the log-densities of particular values of 1000 latent variables.

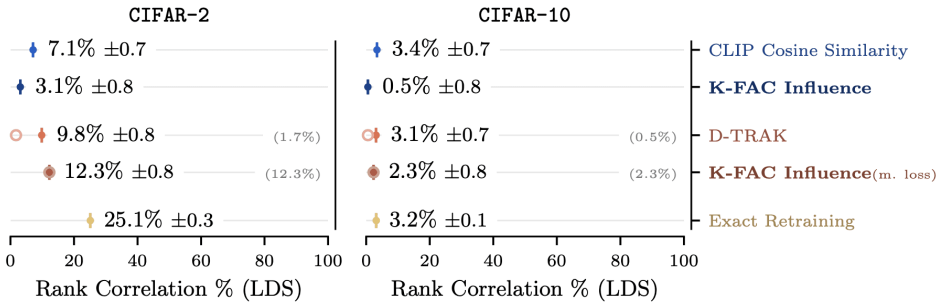


Figure 19. Linear Data-modelling Score (LDS) for the **probability of sampling trajectory**. The plot follows the same format as that of Figures 2a and 2b. Overall, probability of the sampling trajectory appears to be a difficult proxy for the marginal probability of sampling a given example, given that it suffers from the same issues as the ELBO on CIFAR-2 (it's better approximated by the wrong measurement function), and there is extremely little correlation in the measurement across the retrained models on larger datasets (CIFAR-10).

doi.org/10.32388/BOJDXM

Appendix G. Experimental details

In this section, we describe the implementation details for the methods and baselines, as well as the evaluations reported in Section 4.

G.1. Datasets

We focus on the following dataset in this paper:

CIFAR-10 CIFAR-10 is a dataset of small RGB images of size 32×32 [30]. We use 50000 images (the train split) for training.

CIFAR-2 For CIFAR-2, we follow [15] and create a subset of CIFAR-10 with 5000 examples of images only corresponding to classes `car` and `horse`. 2500 examples of class `car` and 2500 examples of class `horse` are randomly subsampled without replacement from among all CIFAR-10 images of that class.

G.2. Models

For all CIFAR datasets, we train a regular Denoising Diffusion Probabilistic Model using a standard U-Net architecture as described for CIFAR-10 in [13]. This U-Net architecture contains both convolutional and attention layers. We use the same noise schedule as described for the CIFAR dataset in [13].

Sampling

We follow the standard DDPM sampling procedure with a full 1000 timesteps to create the generated samples as described by [13]. DDPM sampling usually gives better samples (in terms of visual fidelity) than Denoising Diffusion Implicit Models (DDIM) sampling [31] when a large number of sampling steps is used. As described in Section 2.1, when parameterising the conditionals $p_\theta(x^{(t-1)}|x^{(t)})$ with neural networks as $\mathcal{N}\left(x^{(t-1)}|\mu_{t-1|t,0}\left(x^{(t)}, \epsilon_\theta^t(x^{(t)})\right), \sigma_t^2 I\right)$ we have a choice in how to set the variance hyperparameters $\{\sigma_t^2\}_{t=1}^T$. The σ_t^2 hyperparameters do not appear in the training loss; however, they do make a difference when sampling. We use the "small" variance variant from §3.2 [13], i.e. we set:

$$\sigma_t^2 = \frac{1 - \prod_{t'=1}^{t-1} \lambda_{t'}}{1 - \prod_{t'=1}^{t} \lambda_{t'}}(1 - \lambda_t)$$

G.3. Details on data attribution methods

TRAK

For TRAK baselines, we adapt the implementation of [16][14] to the diffusion modelling setting. When running TRAK, there are several settings the authors recommend to consider: 1) the projection dimension d_{proj} for the

random projections, 2) the damping factor λ, and 3) the numerical precision used for storing the projected gradients. For (1), we use a relatively large projection dimension of 32768 as done in most experiments in [15]. We found that the projection dimension affected the best obtainable results significantly, and so we couldn't get away with a smaller one. We also found that using the default `float16` precision in the TRAK codebase for (3) results in significantly degraded results (see 20, and so we recommend using `float32` precision for these methods for diffusion models. In all experiments, we use `float32` throughout. For the damping factor, we report the sweeps over LDS scores in Figures 8 and 10, and use the best result in each benchmark, as these methods fail drastically if the damping factor is too small. The damping factor reported in the plots is normalised by the dataset size N, to match the definition of the GGN, and to make it comparable with the damping reported for other influence functions methods introduced in this paper. For non-LDS experiments, we use the best damping value from the corresponding LDS benchmark.

CLIP cosine similarity

One of the data attribution baselines used for the LDS experiments is CLIP cosine similarity [29]. For this baseline, we compute the CLIP embeddings [29] of the generated sample and training datapoints, and consider the cosine similarity between the two as the "influence" of that training datapoint on that particular target sample. See [16] for details of how this influence is aggregated for the LDS benchmark. Of course, this computation does not in any way depend on the diffusion model or the measurement function used, so it is a pretty naïve method for estimating influence.

K-FAC

We build on the https://github.com/f-dangel/curvlinops package for our implementation of K-FAC for diffusion models. Except for the ablation in Figure 4, we use the K-FAC expand variant throughout. We compute K-FAC for PyTorch `nn.Conv2d` and `nn.Linear` modules (including in attention), ignoring the parameters in the normalisation layers.

Compression

For all K-FAC influence functions results, we use `int8` quantisation for the query gradients.

Monte Carlo computation of gradients and the GGN for influence functions

Computing the per-example training loss $\ell(\theta, x_n)$ in Section 2.1, the gradients of which are necessary for computing the influence function approximation (Equation (6)), includes multiple nested expectations over diffusion timestep $\tilde{t}$ and noise added to the data $\epsilon^{(t)}$. This is also the case for the $\mathrm{GGN}^{\mathrm{model}}_{\mathcal{D}}$ in Equation (9) and for the gradients $\nabla_\theta \ell(\theta, x_n)$ in the computation of $\mathrm{GGN}^{\mathrm{loss}}_{\mathcal{D}}$ in Equation (11), as well as for the computation of

the measurement functions. Unless specified otherwise, we use the same number of samples for a Monte Carlo estimation of the expectations for all quantities considered. For example, if we use K samples, that means that for the computation of the gradient of the per-example-loss $\nabla_\theta \ell(\theta, x_n)$ we'll sample tuples of $(\tilde{t}, \epsilon^{(\tilde{t})}, x^{(\tilde{t})})$ independently K times to form a Monte Carlo estimate. For $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{model}}$, we explicitly iterate over all training data points, and draw K samples of $(\tilde{t}, \epsilon^{(\tilde{t})}, x_n^{(\tilde{t})})$ for each datapoint. For $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$, we explicitly iterate over all training data points, and draw K samples of $(\tilde{t}, \epsilon^{(\tilde{t})}, x_n^{(\tilde{t})})$ to compute the gradients $\nabla_\theta \ell(\theta, x_n)$ before taking an outer product. Note that, for $\mathrm{GGN}_{\mathcal{D}}^{\mathrm{loss}}$, because we're averaging over the samples before taking the outer product of the gradients, the estimator of the GGN is no longer unbiased. Similarly, K samples are also used for computing the gradients of the measurement function.

For all CIFAR experiments, we use 250 samples throughout for all methods (including all gradient and GGN computations for K-FAC Influence, TRAK, D-TRAK), unless explicitly indicated in the caption otherwise.

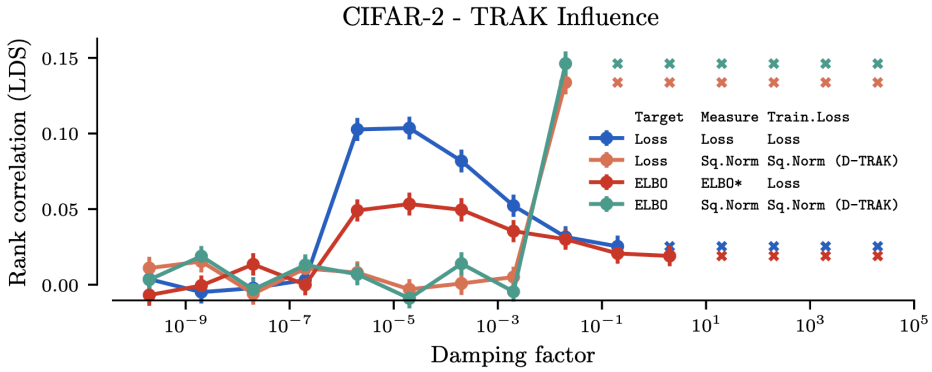


Figure 20. LDS scores on for TRAK (random projection) based influence on CIFAR-2 when using half-precision (`float16`) for influence computations. Compare with Figure 8. NaN results are indicated with $\times$.

G.4. Damping

For all influence function-like methods (including TRAK and D-TRAK), we use damping to improve the numerical stability of the Hessian inversion. Namely, for any method that computes the inverse of the approximation to the Hessian $H \approx \nabla_\theta^2 \mathcal{L}_{\mathcal{D}} = \nabla_\theta^2 \frac{1}{N} \sum \ell(\mathrm{theta}, x_n)$, we add a damping factor λ to the diagonal before inversion:

$$(H + \lambda I)^{-1},$$

where I is a $d_{\mathrm{param}} \times d_{\mathrm{param}}$ identity matrix. This is particularly important for methods where the Hessian approximation is at a high risk of being low-rank (for example, when using the empirical GGN in Equation (11),

doi.org/10.32388/BOJDXM

which is the default setting for TRAK and D-TRAK). For TRAK/D-TRAK, the approximate Hessian inverse is computed in a smaller projected space, and so we add λ to the diagonal directly in that projected space, as done in [15]). In other words, if $P \in \mathbb{R}^{d_{\text{proj}} \times d_{\text{param}}}$ is the projection matrix (see [16] for details), then damped Hessian-inverse preconditioned vector inner products between two vectors $v_1, v_2 \in \mathbb{R}^{d_{\text{param}}}$ (e.g. the gradients in Equation (6)) would be computed as:

$$(Pv_1)^\top (H + \lambda I)^{-1} P v_2.$$

where $H \approx P \nabla^2_\theta \mathcal{L}_\mathcal{D} P^\top \in \mathbb{R}^{d_{\text{proj}} \times d_{\text{proj}}}$ is an approximation to the Hessian in the projected space.

For the "default" values used for damping for TRAK, D-TRAK and K-FAC Influence, we primarily follow recommendations from prior work. For K-FAC Influence, the default is a small damping value 10^{-8} throughout added for numerical stability of inversion, as done in prior work[32]. For TRAK-based methods,[16] recommend using no damping. Hence, we use the lowest numerically stable value of 10^{-9} as the default value throughout.

Note that all damping values reported in this paper are reported as if being added to the GGN for the Hessian of the loss *normalised by dataset size*. This differs from the damping factor in the TRAK implementation (https://github.com/MadryLab/trak), which is added to the GGN for the Hessian of an unnormalised loss ($\sum_n \ell(\theta, x_n)$). Hence, the damping values reported in[15] are larger by a factor of N (the dataset size) than the equivalent damping values reported in this paper.

G.5. LDS Benchmarks

For all LDS benchmarks[16], we sample 100 sub-sampled datasets ($M := 100$ in 13), and we train 5 models with different random seeds ($K := 5$ in 13), each with 50% of the examples in the full dataset, for a total of 500 retrained models for each benchmark. We compute the LDS scores for 200 samples generated by the model trained on the full dataset.

Monte Carlo sampling of measurements

For all computations of the "true" measurement functions for the retrained models in the LDS benchmarks we use 5000 samples to estimate the measurement.

G.6. Retraining without top influences

For the retraining without top influences experiments (Figure 3), we pick 5 samples generated by the model trained on the full dataset, and, for each, train a model with a fixed percentage of most influential examples for that sample removed from the training dataset, using the same procedure as training on the full dataset (with the same number of training steps). We then report the change in the measurement on the sample for which top influences were removed.

Monte Carlo sampling of measurements

Again, for all computations of the "true" measurement functions for the original and the retrained models used for calculating the difference in loss after retraining we use 5000 samples to estimate the measurement.

G.7. Training details

For CIFAR-10 and CIFAR-2 we again follow the training procedure outlined in[13], with the only difference being a shortened number of training iterations. For CIFAR-10, we train for 160000 steps (compared to 800000 in[13]) for the full model, and 80000 steps for the subsampled datasets (410 epochs in each case). On CIFAR-2, we train for 32000 steps for the model trained on the full dataset, and 16000 steps for the subsampled datasets (800 epochs). We train for significantly longer than[15], as we noticed the models trained using their procedure were somewhat significantly undertrained (some per-diffusion-timestep training losses $\ell_t(\theta, x)$ have not converged). We also use a cosine learning-rate schedule for the CIFAR-2 models.

G.8. Handling of data augmentations

In the presentation in Section 2, we ignore for the sake of clear presentation the reality that in most diffusion modelling applications we also apply data augmentations to the data. For example, the training loss $\mathcal{L}_\mathcal{D}$ in Equation (3) in practice often takes the form:

$$\mathcal{L}_\mathcal{D} = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{\tilde{x}_n}\left[\ell(\theta, \tilde{x}_n)\right],$$

where $\tilde{x}_n$ is the data point x_n after applying a (random) data augmentation to it. This needs to be taken into account 1) when defining the GGN, as the expectation over the data augmentations $\mathbb{E}_{\tilde{x}_n}$ can either be considered as part of the outer expectation $\mathbb{E}_z$, or as part of the loss ρ (see 2.2.1), 2) when computing the per-example train loss gradients for influence functions, 3) when computing the loss measurement function.

When computing $\mathrm{GGN}^{\mathrm{model}}_\mathcal{D}$ in Equation (9), we treat data augmentations as being part of the out "empirical data distribution". In other words, we would simply replace the expectation $\mathbb{E}_{x_n}$ in the definition of the GGN with a nested expectation $\mathbb{E}_{x_n}\mathbb{E}_{\tilde{x}_n}$:

$$\mathrm{GGN}^{\mathrm{model}}_\mathcal{D}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{x}_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})},\epsilon^{(\tilde{t})}}\left[\nabla_\theta^\top \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\,(2I)\,\nabla_\theta \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})\right]\right]\right]\right].$$

with $x^{(\tilde{t})}$ now being sampled from the diffusion process $q(x^{(\tilde{t})}|\tilde{x}_n)$ conditioned on the augmented sample $\tilde{x}_n$. The terms changing from the original equation are indicated in yellow. The "Fisher" expression amenable to MC sampling takes the form:

$$F_\mathcal{D}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{x}_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x_n^{(\tilde{t})},\epsilon^{(\tilde{t})}}\mathbb{E}_{\epsilon_{\mathrm{mod}}}\left[g_n(\theta)g_n(\theta)^\top\right]\right]\right]\right],$$

$$\epsilon_{\text{mod}} \sim \mathcal{N}(\epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})}), I),$$

where, again, $g_n(\theta) = \nabla_\theta \|\epsilon_{\text{mod}} - \epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})})\|^2$.

When computing $\text{GGN}_{\mathcal{D}}^{\text{loss}}$ in Equation (11), however, we treat the expectation over daea augmentations as being part of the loss ρ, in order to be more compatible with the implementations of TRAK[16] in prior works that rely on an empirical GGN[15][14].[10] Hence, the GGN in Equation (11) takes the form:

$$\begin{aligned}\text{GGN}_{\mathcal{D}}^{\text{loss}}(\theta) &= \mathbb{E}_{x_n}\left[\nabla_\theta\left[\mathbb{E}_{\tilde{x}_n}\left[\ell(\theta, \tilde{x}_n)\right]\right] \nabla_\theta^\top\left[\mathbb{E}_{\tilde{x}_n}\left[\ell(\theta, \tilde{x}_n)\right]\right]\right] \\ &= \mathbb{E}_{x_n}\left[\nabla_\theta \tilde{\ell}\,(\theta, \tilde{x}_n) \nabla_\theta^\top \tilde{\ell}\,(\theta, \tilde{x}_n)\right],\end{aligned}$$

where $\tilde{\ell}$ is the per-example loss in expectation over data-augmentations. This is how the Hessian approximation is computed both when we're using K-FAC with $\text{GGN}_{\mathcal{D}}^{\text{model}}$ in presence of data augmentations, or when we're using random projections (TRAK and D-TRAK).

When computing the training loss gradient in influence function approximation in equation (5), we again simply replace the per-example training loss $\ell(\theta^\star, x_j)$ with the per-example training loss averaged over data augmentations $\widetilde{\ell}(\theta^\star, x_j)$, so that the training loss $\mathcal{L}_{\mathcal{D}}$ can still be written as a finite sum of per-example losses as required for the derivation of influence functions.

For the measurement function m in Equation (6), we assume we are interested in the log probability of (or loss on) a particular query example in the particular variation in which it has appeared, so we do not take data augmentations into account in the measurement function.

Lastly, since computing the training loss gradients for the influence function approximation for diffusion models usually requires drawing MC samples anyways (e.g. averaging per-diffusion timestep losses over the diffusion times $\widetilde{t}$ and noise samples $\epsilon^{(t)}$), we simply report the total number of MC samples per data point, where data augmentations, diffusion time $\widetilde{t}$, etc. are all drawn independently for each sample.

Acknowledgments

We thank Jihao Andreas Lin for useful discussions on compression, and help with implementation of quantisation and SVD compression. We would also like to thank Kristian Georgiev for sharing with us the diffusion model weights used for analysis in[14], and Felix Dangel for help and feedback on implementing extensions to the curvlinops package used for the experiments in this paper. Richard E. Turner is supported by Google, Amazon, ARM, Improbable, EPSRC grant EP/T005386/1, and the EPSRC Probabilistic AI Hub (ProbAI, EP/Y028783/1).

Footnotes

[1] Note that the two random variables $x^{(t)}, \epsilon^{(t)}$ are deterministic functions of one-another.

[2] Equivalently, a weighted sum of per-timestep negative log-likelihoods $-\log p_\theta(x^{t-1}|x^{(t)})$.

[3] h_z is typically required to be convex to guarantee the resulting GGN is a positive semi-definite (PSD) matrix. A valid non-PSD approximation to the Hessian can be formed with a non-convex h_z as well; all the arguments about the exactness of the GGN approximation for a linear f_z would still apply. However, the PSD property helps with numerical stability of the matrix inversion, and guarantees that the GGN will be invertible if a small damping term is added to the diagonal.

[4] Generally, $\mathbb{E}_{x_n}$ might also subsume the expectation over data augmentations applied to the training data points (see Appendix G.8 for details on how this is handled).

[5] This is because the Hessian of an ℓ_2-loss w.r.t. the model output is a multiple of the identity matrix.

[6] For the sake of a simpler presentation this does not take potential weight sharing into account.

[7] Unless the trained model satisfies very specific "consistency" constraints (Theorem 2[20]).

[8] We can rescale the latent variables $x^{(t)}$ without affecting the marginal distribution $p_\theta(x)$, but changing the probability density of any particular trajectory. Better LDS results can sometimes be obtained when looking at validation examples [15], but diffusion models are used primarily for sampling, so attributing generated samples is of primary practical interest.

[9] Note that, unlike[15], we only change the measurement function for a proxy in the influence function approximation, keeping the Hessian approximation and training loss gradient in Equation (6) the same.

[10] The implementations of these methods store the (randomly projected) per-example training loss gradients for each example before computing the Hessian approximation. Hence, unless data augmentation is considered to be part of the per-example training loss, the number of gradients to be stored would be increased by the number of data augmentation samples taken.

References

1. ^*Schuhmann C, Beaumont R, Vencu R, Gordon C, Wightman R, Cherti M, Coombes T, Katta A, Mullis C, Wortsman M, Schramowski P, Kundurthy S, Crowson K, Schmidt L, Kaczmarczyk R, Jitsev J (2022). "LAION-5B: An open large-scale dataset for training next generation image-text models". arXiv. Available from: https://arxiv.org/abs/2210.08402.*
2. a, b*Saveri J, Butterick M. "Image Generator Litigation". https://imagegeneratorlitigation.com/, 2023. Accessed: 2024-07-06.*
3. ^*Saveri J, Butterick M (2023). "Language Model Litigation". https://llmlitigation.com/. Accessed: 2024-07-06.*

4. ^Koh PW, Liang P. Understanding black-box predictions via influence functions. In: Precup D, Teh YW, editors. Proceedings of the 34th International Conference on Machine Learning. Proceedings of Machine Learning Research. 2017;70:1885-1894. Available from: https://proceedings.mlr.press/v70/koh17a.html.
5. ^Bae J, Ng N, Lo A, Ghassemi M, Grosse R. If Influence Functions are the Answer, Then What is the Question? arXiv [Internet]. 2022 Sep [cited 2023 Jun 12]; Available from: https://arxiv.org/abs/2209.05364.
6. ^Schraudolph NN (2002). "Fast Curvature Matrix-Vector Products for Second-Order Gradient Descent". Neural computation. **14** (7).
7. a, b, c, dMartens J (2020). "New insights and perspectives on the natural gradient method". JMLR. **21** (146).
8. a, bHeskes T (2000). "On 'natural' learning and pruning in multilayered perceptrons". Neural Computation. **12** (4).
9. a, bMartens J, Grosse R. "Optimizing neural networks with Kronecker-factored approximate curvature". In: ICML; 2015.
10. a, bKwon Y, Wu E, Wu K, Zou J. "DataInf: Efficiently Estimating Data Influence in LoRA-tuned LLMs and Diffusion Models." In: The Twelfth International Conference on Learning Representations; 2023 Oct.
11. a, b, cGrosse R, Martens J (2016). "A Kronecker-factored approximate Fisher matrix for convolution layers". In: ICML.
12. a, b, c, d, e, f, gEschenhagen R, Immer A, Turner RE, Schneider F, Hennig P (2023). "Kronecker-Factored Approximate Curvature for Modern Neural Network Architectures". In: NeurIPS.
13. a, b, c, d, e, f, g, h, i, j, k, l, m, nHo J, Jain A, Abbeel P (2020). "Denoising Diffusion Probabilistic Models". In: Advances in Neural Information Processing Systems, vol. 33, pp. 6840-6851. Curran Associates, Inc.
14. a, b, c, d, e, f, gGeorgiev K, Vendrow J, Salman H, Park SM, Madry A. The Journey, Not the Destination: How Data Guides Diffusion Models. arXiv. December 2023. Available from: arXiv:2312.06205.
15. a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, pZheng X, Pang T, Du C, Jiang J, Lin M. Intriguing Properties of Data Attribution on Diffusion Models. arXiv. 2024 Mar. doi:10.48550/arXiv.2311.00500.
16. a, b, c, d, e, f, g, hPark SM, Georgiev K, Ilyas A, Leclerc G, Madry A. TRAK: Attributing Model Behavior at Scale. arXiv. April 2023. doi:10.48550/arXiv.2303.14186.
17. a, bSohl-Dickstein J, Weiss EA, Maheswaranathan N, Ganguli S. Deep Unsupervised Learning Using Nonequilibrium Thermodynamics. 2015 Nov. doi:10.48550/arXiv.1503.03585.
18. a, b, cTurner RE, Diaconu CD, Markou S, Shysheya A, Foong AYK, Mlodozeniec B. "Denoising Diffusion Probabilistic Models in Six Simple Steps". 2024. Available from: arXiv:2402.04384.
19. ^Song Y, Ermon S (2020). "Generative Modeling by Estimating Gradients of the Data Distribution". arXiv. arXiv:1907.05600 [cs.LG].

20. a, b, cSong Y, Sohl-Dickstein J, Kingma DP, Kumar A, Ermon S, Poole B. Score-Based Generative Modeling throug h Stochastic Differential Equations. 2021 Feb. doi:10.48550/arXiv.2011.13456.
21. a, bSong Y, Durkan C, Murray I, Ermon S. Maximum Likelihood Training of Score-Based Diffusion Models. arXiv. October 2021. Available from: arXiv:2101.09258.
22. ^Kingma DP, Salimans T, Poole B, Ho J. Variational Diffusion Models. arXiv. April 2023. Available from: https://ar xiv.org/abs/2107.00630.
23. a, bKrantz SG, Parks HR. The Implicit Function Theorem. Boston, MA: Birkhäuser; 2003. ISBN 978-1-4612-6593-1 978-1-4612-0059-8. doi:10.1007/978-1-4612-0059-8.
24. ^Kunstner F, Balles L, Hennig P (2019). "Limitations of the Empirical Fisher Approximation for Natural Gradient Descent". In: NeurIPS.
25. ^Amari S. Natural gradient works efficiently in learning. Neural computation. **10** (2), 1998.
26. a, b, c, d, e, fGrosse R, Bae J, Anil C, Elhage N, Tamkin A, Tajdini A, Steiner B, Li D, Durmus E, Perez E, Hubinger E, L ukošiūtė K, Nguyen K, Joseph N, McCandlish S, Kaplan J, Bowman SR. Studying Large Language Model Generaliza tion with Influence Functions. arXiv. 2023 Aug. arXiv: 2308.03296.
27. ^Bernacchia A, Lengyel M, Hennequin G (2018). "Exact natural gradient in deep linear networks and its applicati on to the nonlinear case". In: NeurIPS.
28. ^Dasgupta S, Gupta A (2003). "An elementary proof of a theorem of Johnson and Lindenstrauss". Random Structu res & Algorithms. **22** (1): 60–65. doi:10.1002/rsa.10073.
29. a, b, cRadford A, Kim JW, Hallacy C, Ramesh A, Goh G, Agarwal S, Sastry G, Askell A, Mishkin P, Clark J, Krueger G, Sutskever I (2021). "Learning Transferable Visual Models From Natural Language Supervision". arXiv. Available f rom: https://arxiv.org/abs/2103.00020.
30. ^Krizhevsky A. Learning Multiple Layers of Features from Tiny Images. Technical report. University of Toronto; 20 09. Available from: http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf.
31. ^Song J, Meng C, Ermon S. "Denoising Diffusion Implicit Models". arXiv. October 2022. doi:10.48550/arXiv.2010.0 2502.
32. ^Bae J, Lin W, Lorraine J, Grosse R (2024). "Training Data Attribution via Approximate Unrolled Differentiation". arXiv. https://arxiv.org/abs/2405.12186.

Declarations

Funding: Richard E. Turner is supported by Google, Amazon, ARM, Improbable, EPSRC grant EP/T005386/1, and the EPSRC Probabilistic AI Hub (ProbAI, EP/Y028783/1).

Potential competing interests: No potential competing interests to declare.