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]University of Cambridge [2]Max Planck Institute for Intelligent Systems, Tübingen
[3]University of Toronto [4]Vector Institute [5]ETH Zurich
[6]MILA [7]The Alan Turing Institute

Abstract

Diffusion models have led to significant advancements in generative modelling. Yet their widespread adoption poses challenges regarding data attribution and interpretability. 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 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.

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 [26], 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" [23], which has caused a flurry of high-profile lawsuits [23, 24]. 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 [13, 2] 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.

[*]Correspondence to: bkm28@cam.ac.uk

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

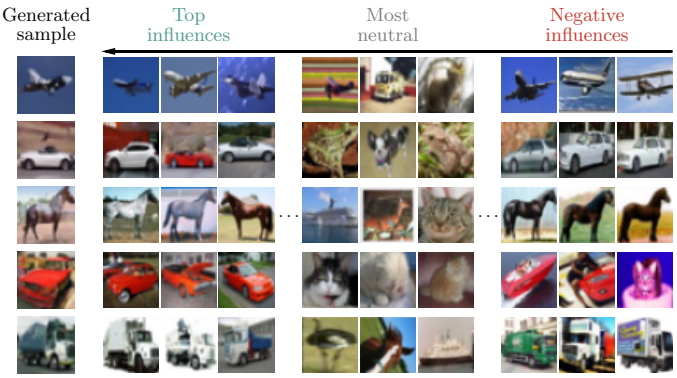


Figure 1: Most influential training data points as identified by K-FAC Influence Functions for samples generated by a denoising diffusion probabilistic model trained on `CIFAR-10`. The top influences are those whose omission from the training set is predicted to most increase the loss of the generated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

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 3). 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, 25, 19]. 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.

In this work, we describe a scalable approach to influence function-based data attribution in diffusion models, using a Kronecker-Factored Approximate Curvature [K-FAC, 10, 20] approximation of GGNs as Hessian approximations. We articulate a design space based on influence functions, unify previous methods for data attribution in diffusion models [7, 30] 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 [21] or retraining without top influences. Finally, we discuss interesting empirical observations that challenge our current understanding of influence functions in the context of diffusion models.

2 Background

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

2.1 Diffusion Models

Diffusion models are a class of probabilistic generative models that fit a model $p_\theta(x)$ parameterised by parameters $\theta \in \mathbb{R}^{d_{\texttt{param}}}$ to approximate a training data distribution $q(x)$, with the primary aim being to sample new data $x \sim p_\theta(\cdot)$ [27, 11, 29]. 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 with hyperparameters λ_t set so that the law of $x^{(T)}$ approximately matches a standard Gaussian distribution $\mathcal{N}(0, I)$. The more detailed overview is presented in Appendix A.

In the setting described above, the diffusion model training loss for a given timestep $\ell_t(\theta, x^{(0)})$ simplifies to $\mathbb{E}_{x^{(t)}, \epsilon^{(t)}}\left[\|\epsilon^{(t)} - \epsilon_\theta^t(x^{(t)})\|^2\right]$ – a mean squared error loss between the noise $\epsilon^{(t)}$ added

to the data, and a prediction $\epsilon_\theta^t(x^{(t)})$ for the noise that has been added to the data. This leads to a training loss ℓ for the diffusion model that is a sum of per-diffusion timestep training losses:

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

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

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

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 (1) when some training examples $(x_j)_{j\in\mathcal{I}}$, $\mathcal{I} = \{i_1, \ldots, i_M\} \subseteq [N]$, 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:

$$r_{-\mathcal{I}}(\varepsilon) = \arg\min_\theta \frac{1}{N}\sum_{n=1}^N \ell(\theta, x_n) - \varepsilon \sum_{j\in\mathcal{I}} \ell(\theta, x_j) \tag{2}$$

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

Under suitable assumptions (see Appendix B), by the Implicit Function Theorem [15], 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 $\varepsilon = 0$:

$$\begin{aligned} r_{-\mathcal{I}}(\varepsilon) &= r_{-\mathcal{I}}(0) &&+ \left.\frac{dr_{-\mathcal{I}}(\varepsilon')}{d\varepsilon'}\right|_{\varepsilon'=0} \varepsilon &&+ o(\varepsilon) \\ &= \theta^\star(\mathcal{D}) &&+ \sum_{j\in\mathcal{I}} \left(\nabla^2_{\theta^\star}\mathcal{L}_\mathcal{D}(\theta^\star)\right)^{-1} \nabla_{\theta^\star}\ell(\theta^\star, x_j)\varepsilon &&+ o(\varepsilon), \end{aligned} \tag{3}$$

as $\varepsilon \to 0$. See Appendix B 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 $m(\theta^\star(\mathcal{D}), x')$ 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 $\theta^\star(\mathcal{D})$ — $m(\theta, x') = m(\theta^\star, x') + \nabla^\intercal_{\theta^\star} m(\theta^\star, x')(\theta - \theta^\star) + o\left(\|\theta - \theta^\star\|_2\right)$ — and combine it with Equation (3) to get a simple linear estimate of the change in the measurement function:

$$m(r_{-\mathcal{I}}(\varepsilon), x') = m(\theta^\star, x') + \sum_{j\in\mathcal{I}} \nabla^\intercal_{\theta^\star} m(\theta^\star, x') \left(\nabla^2_{\theta^\star}\mathcal{L}_\mathcal{D}(\theta^\star)\right)^{-1} \nabla_{\theta^\star}\ell(\theta^\star, x_j)\varepsilon + o(\varepsilon). \tag{4}$$

2.2.1 Generalised Gauss-Newton matrix

Computing the influence function approximation in Equation (3) requires inverting the Hessian $\nabla^2_\theta \mathcal{L}_\mathcal{D}(\theta) \in \mathbb{R}^{d_{\text{param}} \times d_{\text{param}}}$. 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 $\mathcal{L}(\theta) = \mathbb{E}_z\left[\rho(\theta, z)\right]$, 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

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

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 [19]. 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 $\mathrm{GGN}^{\texttt{model}}$. The GGN with this split is exact for linear neural networks (or when the model has zero residuals on the training data) [19].

$$\begin{aligned} f_z &\coloneqq \text{mapping from parameters to model output} \\ h_z &\coloneqq \text{loss function (e.g. } \ell_2\text{-loss)} \end{aligned} \quad \rightarrow \mathrm{GGN}^{\texttt{model}}(\theta) \tag{5}$$

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

$$\begin{aligned} f_z &\coloneqq \rho(\cdot, z) \\ h_z &\coloneqq \mathrm{id} \end{aligned} \quad \rightarrow \mathrm{GGN}^{\texttt{loss}}(\theta) = \mathbb{E}_z\left[\nabla_\theta \rho(\theta, z) \nabla_\theta^\intercal \rho(\theta, z)\right]. \tag{6}$$

This is also called the empirical Fisher [17]. Note that $\mathrm{GGN}^{\texttt{loss}}$ 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 $\mathrm{GGN}^{\texttt{model}}$, for which only the mapping from the parameters to the model output needs to be (locally) linear. Hence, we might prefer to use $\mathrm{GGN}^{\texttt{model}}$ 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 [7, 30] 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. We will describe how to formulate different GGN approximations for this setting.

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:

$$\begin{aligned} f_z &\coloneqq \epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})}) \\ h_z &\coloneqq \|\epsilon^{(\tilde{t})} - \cdot\|^2 \end{aligned} \quad \rightarrow \mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})}, \epsilon^{(\tilde{t})}}\left[\nabla_\theta^\intercal \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right)(2I)\nabla_\theta \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right)\right]\right]\right], \tag{7}$$

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 $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ as

$$\mathrm{F}_{\mathcal{D}}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x_n^{(\tilde{t})}}\left[\mathbb{E}_{\epsilon_{\texttt{mod}}}\left[g_n(\theta) g_n(\theta)^\intercal\right]\right]\right]\right], \qquad \epsilon_{\texttt{mod}} \sim \mathcal{N}\left(\epsilon_\theta^{\tilde{t}}\left(x_n^{(\tilde{t})}\right), I\right) \tag{8}$$

where $g_n(\theta) = \nabla_\theta \|\epsilon_{\texttt{mod}} - \epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})})\|^2 \in \mathbb{R}^{d_{\texttt{param}}}$; see Appendix C for the derivation. This formulation lends itself to a Monte Carlo approximation, since we can now compute gradients using auxiliary targets $\epsilon_{\texttt{mod}}$ sampled from the model's output distribution, as shown in Equation (8). $\mathrm{F}_{\mathcal{D}}$ can be interpreted as a kind of Fisher information matrix [1, 19], but it is not the Fisher for the marginal model distribution $p_\theta(x)$.

Option 2. Analogously to Equation (6), 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:

$$\begin{aligned} f_z &:= \ell(\cdot, x_n) \\ h_z &:= \mathrm{id} \end{aligned} \quad \rightarrow \mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}(\theta) = \mathbb{E}_{x_n}\left[\nabla_\theta \ell(\theta, x_n) \nabla_\theta^\intercal \ell(\theta, x_n)\right], \tag{9}$$

where $\ell(\theta, x)$ is the diffusion training loss defined in Equation (11). This Hessian approximation $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ turns out to be equivalent to the ones considered in the previous works on data attribution for diffusion models [7, 30, 18]. In contrast, in this work, we opt for $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ in Equation (7), or equivalently $\mathrm{F}_{\mathcal{D}}$, since it is arguably a better-motivated approximation of the Hessian than $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ (c.f. Section 2.2.1).

In Zheng et al. [30], 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 $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ in Equation (7). Hence, in our method substituting different targets would not affect the Hessian approximation. In Zheng et al. [30], replacing the targets only makes a difference to the Hessian approximation because they use $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ (an empirical Fisher) to approximate the Hessian.

K-FAC for diffusion models We further approximate the GGN with a K-FAC matrix, the details of which we describe in Appendix D.1.

3.2 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 and the gradients thereof is challenging, however. Hence, in this work, we consider a couple of proxies:

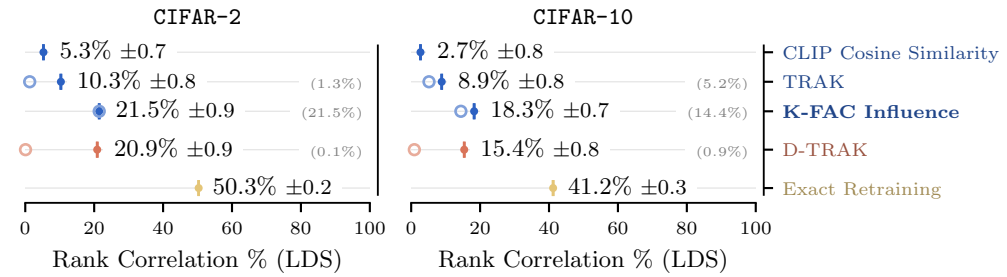
1. **Loss.** Approximate $\log p_\theta(x)$ with the diffusion loss $\ell(\theta, x)$ in Equation (11) on that particular example. This corresponds to the ELBO with reweighted per-timestep loss terms (see Figure 17).
2. **Probability of sampling trajectory.** If the entire sampling trajectory $x^{(0:T)}$ 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 [11, eq. (5)].

4 Experiments

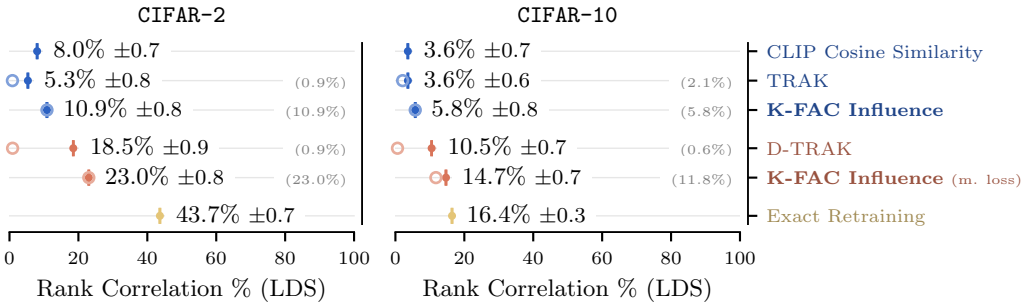
Evaluating Data Attribution. To evaluate the proposed data attribution methods, we primarily focus on two metrics: *Linear Data Modelling Score* (LDS) and *retraining without top influences*. These metrics are described in Appendix E. In all experiments, we look at measurements on samples generated by the model trained on $\mathcal{D}$. We primarily focus on Denoising Diffusion Probabilistic Models (DDPM) [11] throughout. The LDS results are described below, whereas the retraining without top influences results are shown in Appendix J. An ablation over the Hessian approximation is shown in Appendix K.

Baselines We compare influence functions with K-FAC and $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ (MC-Fisher; Equation (8)) as the Hessian approximation (K-FAC Influence) to TRAK as formulated for diffusion models in [7, 30]. We describe the remaining baselines in Appendix M.1.

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 H), as well as for "default" values following recommendations in previous work (see Appendix M.5). TRAK and D-TRAK appear to be more sensitive to tuning the damping factor than K-FAC Influence. They often



(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 ●. Where applicable, we plot results for both the best Hessian-approximation damping value with ● and a "default" damping value with ○. 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 ●. 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.

don't perform at all if the damping factor is too small, and take a noticeable performance hit if the damping factor is not tuned to the problem or method (see Figures 6 and 8 in Appendix H). 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 5 and 7).

One peculiarity in the LDS results, similar to the findings in [30], is that substituting the loss measurement for the ELBO when predicting changes in ELBO actually works better than using the correct measurement (see Figure 2b). We discuss this, and related challenges, in Appendix F.

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. Our proposed method with recommended design choices improves performance compared 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 18), 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. 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.

References

[1] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2), 1998.

[2] Juhan Bae, Nathan Ng, Alston Lo, Marzyeh Ghassemi, and Roger Grosse. If Influence Functions are the Answer, Then What is the Question?, September 2022.

[3] Juhan Bae, Wu Lin, Jonathan Lorraine, and Roger Grosse. Training data attribution via approximate unrolled differentiation, 2024. URL https://arxiv.org/abs/2405.12186.

[4] Alberto Bernacchia, Mate Lengyel, and Guillaume Hennequin. Exact natural gradient in deep linear networks and its application to the nonlinear case. In *NeurIPS*, 2018.

[5] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, January 2003. ISSN 1042-9832, 1098-2418. doi: 10.1002/rsa.10073.

[6] Runa Eschenhagen, Alexander Immer, Richard E. Turner, Frank Schneider, and Philipp Hennig. Kronecker-Factored Approximate Curvature for modern neural network architectures. In *NeurIPS*, 2023.

[7] Kristian Georgiev, Joshua Vendrow, Hadi Salman, Sung Min Park, and Aleksander Madry. The Journey, Not the Destination: How Data Guides Diffusion Models, December 2023.

[8] Roger Grosse and James Martens. A Kronecker-factored approximate Fisher matrix for convolution layers. In *ICML*, 2016.

[9] Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, Evan Hubinger, Kamilė Lukošiūtė, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying Large Language Model Generalization with Influence Functions, August 2023.

[10] Tom Heskes. On "natural" learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4), 2000.

[11] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising Diffusion Probabilistic Models. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6840–6851. Curran Associates, Inc., 2020.

[12] Zahra Kadkhodaie, Florentin Guth, Eero P. Simoncelli, and Stéphane Mallat. Generalization in diffusion models arises from geometry-adaptive harmonic representations, April 2024.

[13] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1885–1894. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/koh17a.html.

[14] Pang Wei Koh, Kai-Siang Ang, Hubert H. K. Teo, and Percy Liang. On the Accuracy of Influence Functions for Measuring Group Effects, November 2019.

[15] Steven G. Krantz and Harold R. Parks. *The Implicit Function Theorem*. Birkhäuser, Boston, MA, 2003. ISBN 978-1-4612-6593-1 978-1-4612-0059-8. doi: 10.1007/978-1-4612-0059-8.

[16] Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. Technical report, University of Toronto, 2009. URL http://www.cs.utoronto.ca/~kriz/learning-features-2009-TR.pdf.

[17] Frederik Kunstner, Lukas Balles, and Philipp Hennig. Limitations of the empirical Fisher approximation for natural gradient descent. In *NeurIPS*, 2019.

[18] Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. DataInf: Efficiently Estimating Data Influence in LoRA-tuned LLMs and Diffusion Models. In *The Twelfth International Conference on Learning Representations*, October 2023.

[19] James Martens. New insights and perspectives on the natural gradient method. *JMLR*, 21(146), 2020.

[20] James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

[21] Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. TRAK: Attributing Model Behavior at Scale, April 2023.

[22] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision, 2021. URL https://arxiv.org/abs/2103.00020.

[23] Joseph Saveri and Matthew Butterick. Image generator litigation. https://imagegeneratorlitigation.com/, 2023. Accessed: 2024-07-06.

[24] Joseph Saveri and Matthew Butterick. Language model litigation. https://llmlitigation.com/, 2023. Accessed: 2024-07-06.

[25] Nicol N Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7), 2002.

[26] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. Laion-5b: An open large-scale dataset for training next generation image-text models, 2022. URL https://arxiv.org/abs/2210.08402.

[27] Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep Unsupervised Learning using Nonequilibrium Thermodynamics, November 2015.

[28] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising Diffusion Implicit Models, October 2022.

[29] Richard E. Turner, Cristiana-Diana Diaconu, Stratis Markou, Aliaksandra Shysheya, Andrew Y. K. Foong, and Bruno Mlodozeniec. Denoising diffusion probabilistic models in six simple steps, 2024.

[30] Xiaosen Zheng, Tianyu Pang, Chao Du, Jing Jiang, and Min Lin. Intriguing Properties of Data Attribution on Diffusion Models, March 2024.

A Background on denoising diffusion probabilistic models

Diffusion models are a class of probabilistic generative models that fit a model $p_\theta(x)$ parameterised by parameters $\theta \in \mathbb{R}^{d_{\texttt{param}}}$ to approximate a training data distribution $q(x)$, with the primary aim being to sample new data $x \sim p_\theta(\cdot)$ [27, 11, 29].

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. 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)** [29]. The marginals $q(x^{(t)}|x^{(0)}) = \mathcal{N}\left(x^{(t)}|\left(\prod_{t'=1}^{t}\lambda_{t'}\right)x^{(0)}, \left(1-\prod_{t'=1}^{t}\lambda_{t'}^2\right)I\right)$ also have a simple Gaussian form, allowing for the augmented samples to be sampled as:

$$x^{(t)} = \prod_{t'=1}^{t}\lambda_t x^{(0)} + \left(1-\prod_{t'=1}^{t}\lambda_{t'}^2\right)^{1/2}\epsilon^{(t)}, \qquad \text{with } \epsilon^{(t)} \sim \mathcal{N}(0, I). \tag{10}$$

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 $t = 1, \dots, T$. We can note that $q(x^{(t-1)}|x^{(t)}, x^{(0)})$ has a Gaussian distribution with mean given by:

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

as in Equation (10). 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 $\mathcal{N}\big(x^{(t-1)}|\mu_{t-1|t,0}\big(x^{(t)}, \epsilon_\theta^t(x^{(t)})\big), \sigma_t^2 I\big)$. 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 [11]. With that parameterisation, the negative expected log-likelihood $\mathbb{E}_{q(x^{t-1},x^{(t)}|x^{(0)})}\left[-\log p(x^{(t-1)}|x^{(t)})\right]$, up to scale and shift independent of θ or $x^{(0)}$, can be written as [11, 29]:[2]

$$\ell_t(\theta, x^{(0)}) = \mathbb{E}_{\epsilon^{(t)},x^{(t)}}\left[\left\|\epsilon^{(t)} - \epsilon_\theta^t\left(x^{(t)}\right)\right\|^2\right] \qquad \begin{aligned} \epsilon^{(t)} &\sim \mathcal{N}(0, I) \\ x^{(t)} &= \prod_{t'=1}^{t}\lambda_t x^{(0)} + \left(1-\prod_{t'=1}^{t}\lambda_{t'}^2\right)^{1/2}\epsilon^{(t)} \end{aligned} \tag{11}$$

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

B Derivation of Influence Functions

In this section, we state the implicit function theorem (Appendix B.1). Then, in Appendix B.2, we introduce the details of how it can be applied in the context of a loss function $\mathcal{L}(\varepsilon, \boldsymbol{\theta})$ 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).

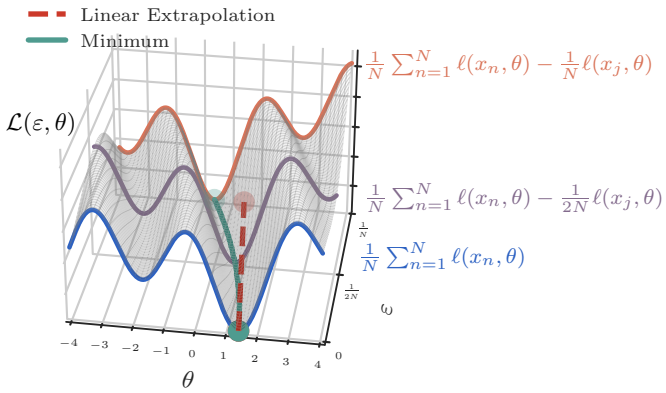


Figure 3: Illustration of the influence function approximation for a 1-dimensional parameter space $\theta \in \mathbb{R}$. Influence funcitons consider the extended loss landscape $\mathcal{L}(\varepsilon, \theta) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{n=1}^{N}\ell(x_n,\theta) - \varepsilon\ell(x_j,\theta)$, 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 $\varepsilon = 0$ (●), we can predicted how the optimal parameters would change when the term $\ell(x_j,\theta)$ is fully removed from the loss (●).

B.1 Implicit Function Theorem

Theorem 1 (Implicit Function Theorem [15]) *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:

$$\frac{\partial g(\mathbf{x})}{\partial \mathbf{x}} = -\left[\nabla_{\mathbf{y}} F(\mathbf{x}, g(\mathbf{x}))\right]^{-1} \nabla_{\mathbf{x}} F(\mathbf{x}, g(\mathbf{x})). \tag{12}$$

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

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

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 Equation (12).

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

Now $g(\boldsymbol{\varepsilon})$ might not necessarily be a minimum of $\boldsymbol{\theta} \mapsto \mathcal{L}(\boldsymbol{\varepsilon}, \boldsymbol{\theta})$. However, by making the further assumption that $\mathcal{L}$ is strictly convex we can ensure that whenever $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\varepsilon}, \boldsymbol{\theta}) = \mathbf{0}$, $\boldsymbol{\theta}$ is a unique minimum, and so $g(\boldsymbol{\varepsilon})$ 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 $(\boldsymbol{\varepsilon}, \boldsymbol{\theta}) \in \mathbb{R}^n \times \mathbb{R}^m$, such that $\theta \mapsto \mathcal{L}(\boldsymbol{\varepsilon}, \boldsymbol{\theta})$ is strictly convex $\forall \boldsymbol{\varepsilon} \in \mathbb{R}^n$. Fix a point $(\boldsymbol{\varepsilon}', \boldsymbol{\theta}^\star)$ such that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}(\boldsymbol{\varepsilon}', \boldsymbol{\theta}^\star) = \mathbf{0}$. Then, by the Implicit Function Theorem applied to $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$, there exists an open set $U \subset \mathbb{R}^n$ containing $\boldsymbol{\theta}^\star$ such that there exists a unique function $g : U \to \mathbb{R}^m$ such that $g(\boldsymbol{\varepsilon}') = \boldsymbol{\theta}^\star$, and $g(\boldsymbol{\varepsilon})$ is the unique minimum of $\theta \mapsto \mathcal{L}(\boldsymbol{\varepsilon}, \boldsymbol{\theta})$ for all $\boldsymbol{\varepsilon} \in U$. Moreover, g is continuously differentiable with derivative:*

$$\frac{\partial g(\boldsymbol{\varepsilon})}{\partial \boldsymbol{\varepsilon}} = -\left[\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}^2}(\boldsymbol{\varepsilon}, g(\boldsymbol{\varepsilon}))\right]^{-1} \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\varepsilon} \partial \boldsymbol{\theta}}(\boldsymbol{\varepsilon}, g(\boldsymbol{\varepsilon})) \tag{13}$$

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

$$\frac{\partial^2 \mathcal{L}}{\partial \varepsilon \partial \boldsymbol{\theta}}(\varepsilon, g(\varepsilon)) = \frac{\partial \mathcal{L}_2}{\partial \boldsymbol{\theta}}(g(\varepsilon)) \tag{14}$$

The above lemma and remark give the result in Equation (3). Namely, in section 2.2:

$$\mathcal{L}(\varepsilon, \boldsymbol{\theta}) = \underbrace{\frac{1}{N}\sum_{i=1}^{N} \ell(\boldsymbol{\theta}, x_i)}_{\mathcal{L}_1} \overbrace{- \frac{1}{M}\sum_{j=1}^{M} \ell(\boldsymbol{\theta}, x_{i_j})\, \varepsilon}^{\mathcal{L}_2} \qquad \overset{\text{eq. (14)}}{\Longrightarrow} \qquad \frac{\partial^2 \mathcal{L}}{\partial \varepsilon \partial \boldsymbol{\theta}} = -\frac{1}{M}\sum_{j=1}^{M} \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\boldsymbol{\theta}, x_{i_j})$$

$$\overset{\text{eq. (13)}}{\Longrightarrow} \qquad \frac{\partial g(\boldsymbol{\varepsilon})}{\partial \boldsymbol{\varepsilon}} = \left[\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}^2}(\boldsymbol{\varepsilon}, g(\boldsymbol{\varepsilon}))\right]^{-1} \frac{1}{M}\sum_{j=1}^{M} \frac{\partial}{\partial \boldsymbol{\theta}} \ell(\boldsymbol{\theta}, x_{i_j})$$

C 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

$$\begin{aligned}
&\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}(\theta) \\
&= \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})}, \epsilon^{(\tilde{t})}}\left[\nabla_\theta^\intercal \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right) \nabla^2_{\epsilon_\theta^{\tilde{t}}(x^{(\tilde{t})})} \left\|\epsilon^{(\tilde{t})} - \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right)\right\|^2 \nabla_\theta \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right)\right]\right] \\
&= \frac{2}{N}\sum_{n=1}^{N} \mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})}, \epsilon^{(\tilde{t})}}\left[\nabla_\theta^\intercal \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right) I \nabla_\theta \epsilon_\theta^{\tilde{t}}\left(x^{(\tilde{t})}\right)\right]\right].
\end{aligned} \tag{15}$$

Note that we used

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

We can substitute I with

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

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

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

we can rewrite:

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

where the last equality follows by the chain rule of differentiation. We can thus rewrite the expression for the GGN in Equation (15) as

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

D Details on approaches to improving scalability of influence functions for diffusion

D.1 K-FAC for diffusion models

While $\mathrm{F}_{\mathcal{D}}(\theta)$ and $\mathrm{GGN}^{\texttt{loss}}_{\mathcal{D}}$ do not require computing full Jacobians or the Hessian of the neural network model, they involve taking outer products of gradients of size $\mathbb{R}^{d_{\texttt{param}}}$, which is still intractable. Kronecker-Factored Approximate Curvature [10, 20, K-FAC] 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 (4) efficient enough to scale to large networks. K-FAC is defined for linear layers, including linear layers with weight sharing like convolutions [8]. 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 [6]. For our recommended method, we choose to approximate the Hessian with a K-FAC approximation of $\mathrm{F}_{\mathcal{D}}$, akin to Grosse et al. [9].

For the parameters θ_l of layer l, the GGN $\mathrm{F}_{\mathcal{D}}$ in Equation (8) is approximated by

$$\mathrm{F}_{\mathcal{D}}(\theta_l) \approx \frac{1}{N^2}\sum_{n=1}^{N} \mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})}_n,\epsilon^{(\tilde{t})}}\left[a^{(l)}_n a^{(l)\intercal}_n\right]\right] \otimes \sum_{n=1}^{N} \mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x^{(\tilde{t})}_n,\epsilon^{(\tilde{t})},\epsilon^{(\tilde{t})}_{\texttt{mod}}}\left[b^{(l)}_n b^{(l)\intercal}_n\right]\right], \tag{16}$$

with $a^{(l)}_n \in \mathbb{R}^{d^l_{\text{in}}}$ being the inputs to the lth layer for data point $x^{(\tilde{t})}_n$ and $b^{(l)}_n \in \mathbb{R}^{d^l_{\text{out}}}$ being the gradient of the ℓ_2-loss w.r.t. the output of the lth layer, and $\otimes$ denoting the Kronecker product.[3] The

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

approximation trivially becomes an equality for a single data point and also for deep linear networks with ℓ_2-loss [4, 6]. We approximate the expectations in Equation (16) 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 Eschenhagen et al. [6]; in the case of convolutional layers this corresponds to Grosse & Martens [8]. 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 5, 7 and 19).

D.2 Gradient compression and query batching

In practice, we recommend computing influence function estimates in Equation (4) by first computing and storing the approximate Hessian inverse, and then iteratively computing the preconditioned inner products $\nabla^\intercal_{\theta^\star} m(\theta^\star, x) \left(\nabla^2_{\theta^\star} \mathcal{L}_\mathcal{D}(\theta^\star)\right)^{-1} \nabla_{\theta^\star} \ell(\theta^\star, x_j)$ for different training datapoints x_j. Following Grosse et al. [9], we use query batching to avoid recomputing the gradients $\nabla_{\theta^\star} \ell(\theta^\star, x_j)$ 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 Grosse et al. [9] (see Appendix G), likely due to the fact that gradients $\nabla_\theta \ell(\theta, x_n)$ are not low-rank in this setting.

E Evaluating Data Attribution

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 $m(\theta^\star(\mathcal{D}), x)$ 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:

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

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, Park et al. [21] propose to use an ensemble average measurement after retraining as the "oracle" target:

$$\text{LDS} = \text{spearman}\left[\left(a(\mathcal{D}, \tilde{\mathcal{D}}_i, x)\right)_{i=1}^M ; \left(\frac{1}{K}\sum\nolimits_{k=1}^K m(\tilde{\theta}^\star_k(\tilde{\mathcal{D}}_i), x)\right)_{i=1}^M\right], \tag{17}$$

where $\tilde{\theta}^\star_k(\mathcal{D}') \in \mathbb{R}^{d_\text{param}}$ 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 $m(\theta^\star(\mathcal{D}'), x)$ 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 $m(\theta^\star(\mathcal{D}'), x)$ relative to $m(\theta^\star(\mathcal{D}), x)$ (measurement by the model trained on the full dataset $\mathcal{D}$).

F Potential challenges to use of influence functions for diffusion models

One peculiarity in the LDS results, similar to the findings in [30], 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)").[4] To

[4]Note that, unlike Zheng et al. [30], we only change the measurement function for a proxy in the influence function approximation, keeping the Hessian approximation and training loss gradient in Equation (4) the same.

try and better understand the properties of influence functions, in this section we perform multiple ablations and report different interesting phenomena that give some insight into the challenges of using influence functions in this setting.

As illustrated in Figure 17, 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 9, 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 13 and 14 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 (11)).

Lastly, although ELBO is perhaps the measurement with the most direct link to the marginal probability of sampling a particular example, we find some peculiarities on the diffusion modelling tasks considered. The below observation in particular puts the usefulness of estimating the change in ELBO for data attribution into question:

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

As illustrated in Figure 15, 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 one sample is more likely to be generated than another by one model (as measured by ELBO), and is less likely to be generated than another by a different model trained on a different random subset of the data. Our observation mirrors that of Kadkhodaie et al. [12] who found that, if diffusion models are trained on non-overlapping subsets of data of sufficient size, they generate near-identical images when sampling with the same noise. This suggests that Observation 3 is not necessarily a deficiency of the ELBO measurement as a proxy for marginal log-probability; the different models are in fact learning nearly identical distributions.

G Gradient compression ablation

In Figure 4, 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 Grosse et al. [9]. 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 Grosse et al. [9]. 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 (11) over multiple noise

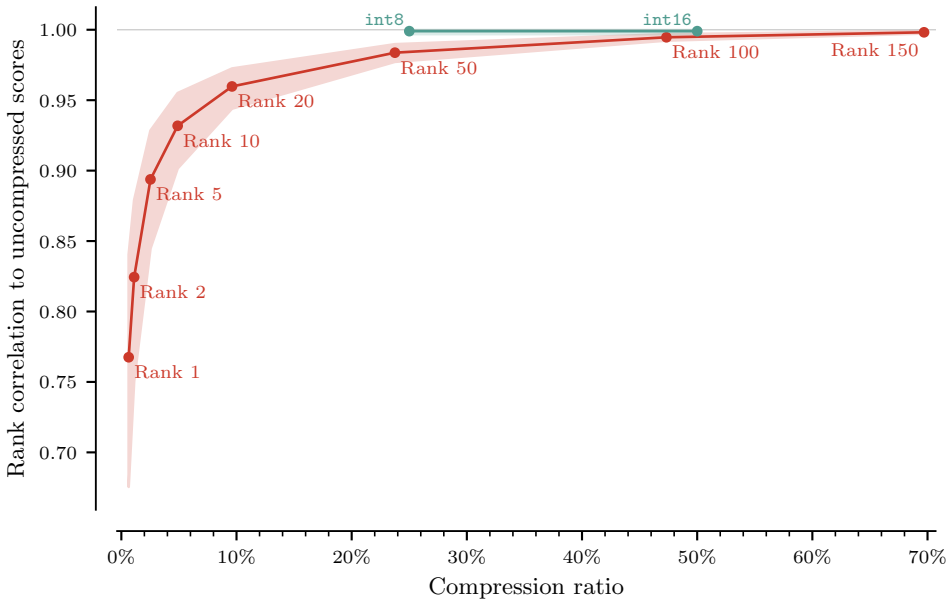


Figure 4: Comparison of gradient compression methods for the influence function approximation.

samples $\epsilon^{(t)}$ and multiple diffusion timesteps. We use 8 bit quantisation with query gradient batching [9] for all KFAC experiments throughout this work.

H 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 5 to 8. 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).

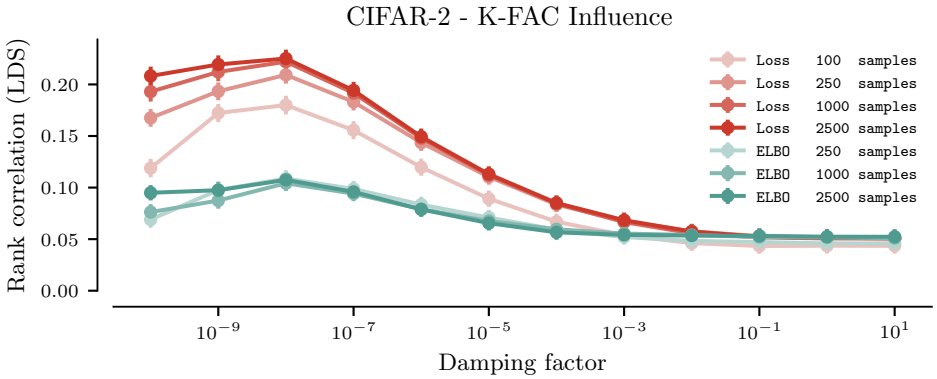


Figure 5: Effect of damping on the LDS scores for **K-FAC influence** on `CIFAR-2`. In this plot, K-FAC GGN approximation was always computed with 1000 samples, and the number of samples used for computing a Monte Carlo estimate of the training loss/measurement gradient is indicated on the legend.

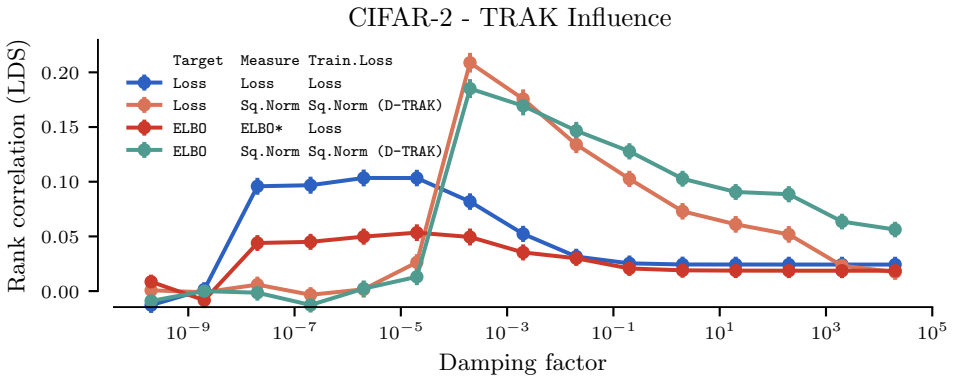


Figure 6: Effect of damping on the LDS scores for TRAK (random projection) based influence on `CIFAR-2`. 250 samples were used for Monte Carlo estiamtion of all quantities (GGN and the training loss/measurement gradients). In the legend: `Target` indicates what measurement we're trying to predict the 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.

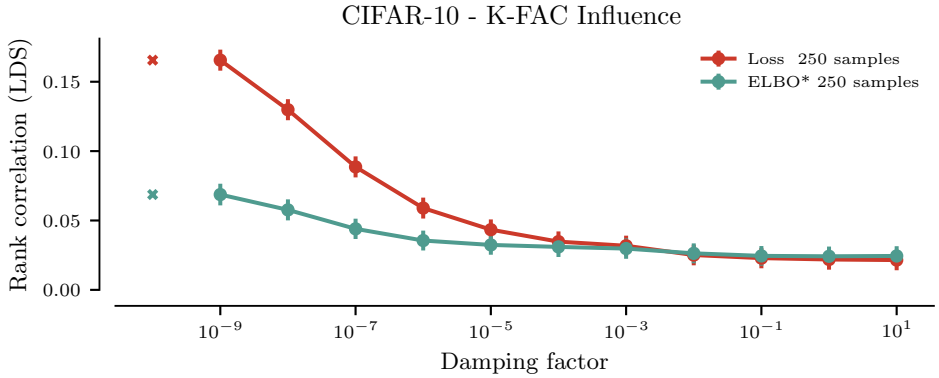


Figure 7: Effect of damping on the LDS scores for K-FAC based influence on `CIFAR-10`. 100 samples were used for computing the K-FAC GGN approximation, and 250 for computing a Monte Carlo estimate of the training loss/measurement gradients. $\times$ indicates a `NaN` result (the computation was not sufficiently numerically stable with that damping factor).

I Empirical ablations for challenges to use of influence functions for diffusion models

In this section, we describe the results for the observations discussed in Appendix F.

Observation 1 is based on Figures 9 and 10. Figure 9 shows the LDS scores on `CIFAR-2` when attributing per-timestep diffusion losses ℓ_t (see Equation (11)) 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 (4)). Figure 10 is a counter-equivalent to Figure 14 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.

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 (4) 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

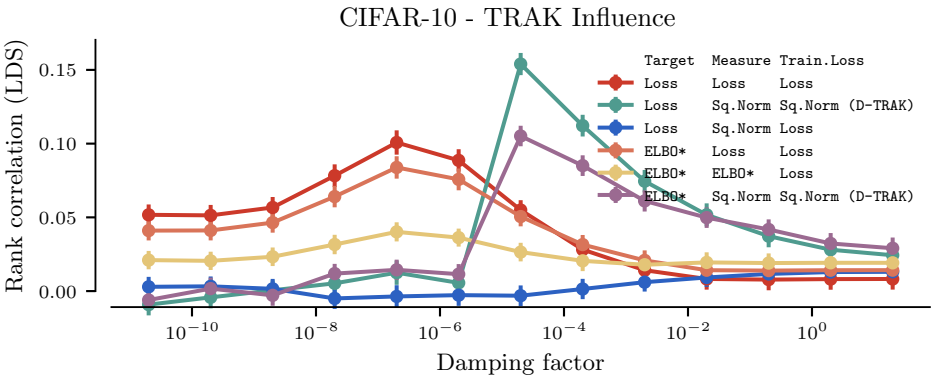


Figure 8: Effect of damping on the LDS scores for TRAK (random projection) based influence on `CIFAR-10`. 250 samples were used for Monte Carlo estiamtion of all quantities (GGN and the training loss/measurement gradients). In the legend: `Target` indicates what measurement we're trying to predict the 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.

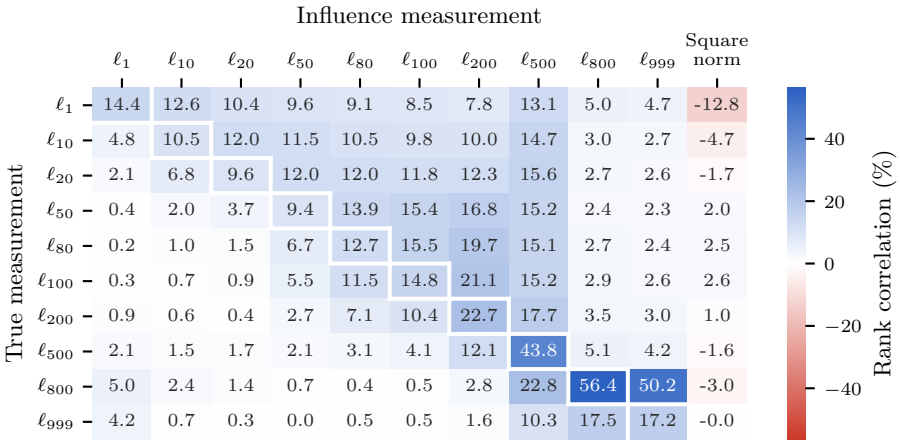


Figure 9: Rank correlation (LDS scores) between influence function estimates with different measurement functions and different true measurements `CIFAR-2`. The plot shows how well different per-timestep diffusion losses ℓ_t work as measurement functions in the influence function approximation, when trying to approximate changes in the actual measurements when retraining a model.

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 9 and 10, 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 11 and 12. 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 12. We also observe that in this setting, influence functions fail completely to predict changes in ℓ_t with the correct measurement function for $t \leq 200$.

Observation 2 Figure 13 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

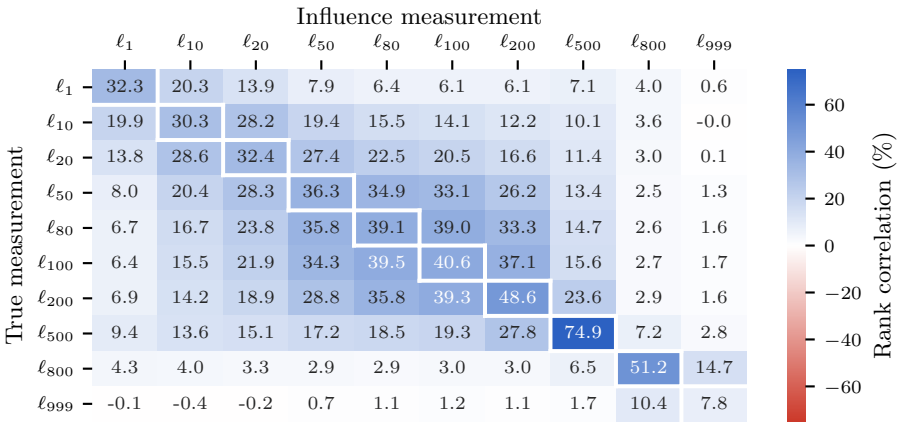


Figure 10: Rank correlation between true measurements for losses at different diffusion timesteps on CIFAR-2.

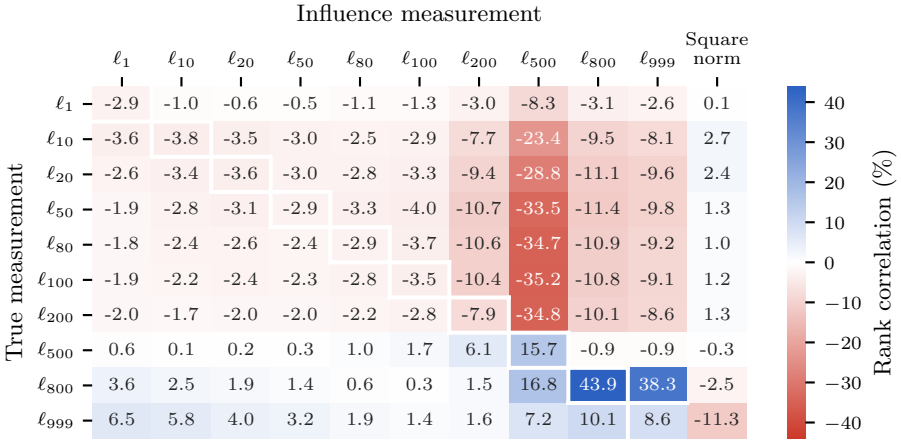


Figure 11: Rank correlation (LDS scores) between influence function estimates with different measurement functions and different true measurements CIFAR-2, but with the retrained models trained on the full dataset with a random subset of examples having a **down-weighted contribution to a training loss by a factor of** $\times 0.5$.

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 (11)) for earlier timesteps t, which can be seen in Figure 14. 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.

Figures 13 and 14 also shows that influence functions tend to overestimate the *magnitude* of the change in measurement after removing the training data points. This is in contrast to the observation in [14] in the supervised setting, where they found that influence functions tend to *underestimate* the magnitude of the change in the measurement. There are many plausible reasons for this, ranging from the choice of the Hessian approximation ([14] compute exact inverse-Hessian-vector products, whereas we approximate the GGN), to the possible "stability" of the learned distribution in diffusion models even when different subsets of data are used for training (Observation 3 and [12]).

Observation 3 Lastly, the observations that the ELBO measurements remain essentially constant for models trained on different subsets of data is based on Figure 15. There, we plot the values of the ELBO measurement for different pairs of models trained on different subsets of data, where we find

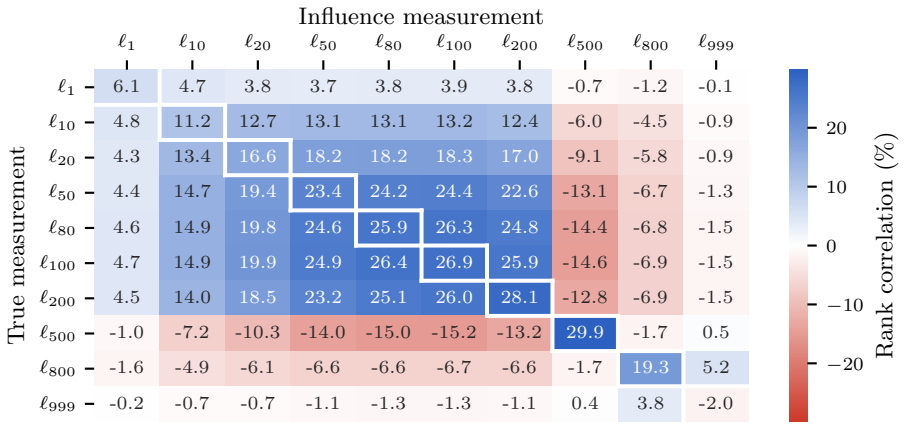


Figure 12: Rank correlation between true measurements for losses at different diffusion timesteps on `CIFAR-2`, but with the retrained models trained on the full dataset with a random subset of examples having a **down-weighted contribution to a training loss by a factor of** $\times 0.5$.

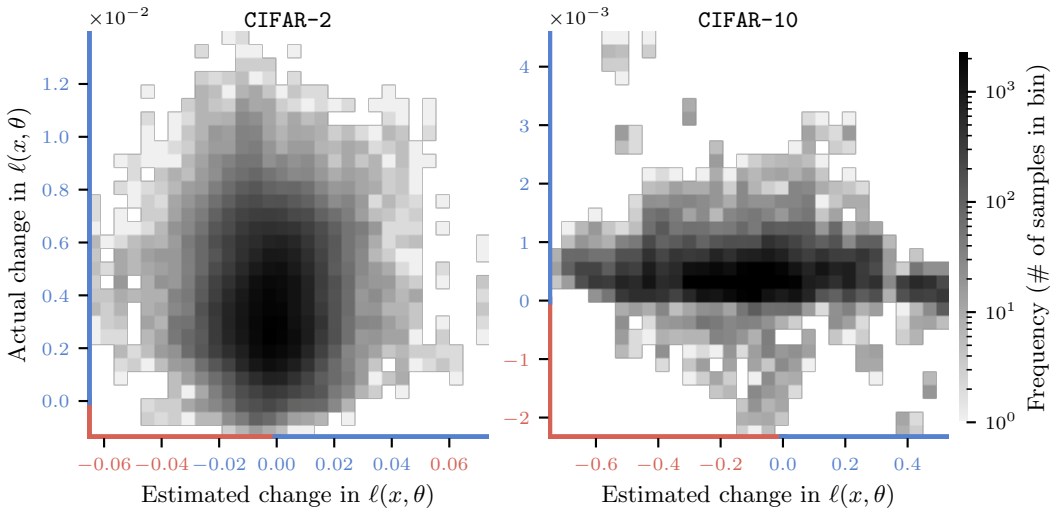


Figure 13: Change in diffusion loss ℓ in Section 2.1 when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for measurements $\ell(x, \theta)$ for 50 samples x generated 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.

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 $5\times$ 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.

Interestingly, the observation does to an extent hold for the simple diffusion loss as well (see Figure 16). For `CIFAR-10`, the correlation is again close to 100% among the retrained models, but for `CIFAR-2` it's substantially lower. This is consistent with the results in [12, Figure 2], where the

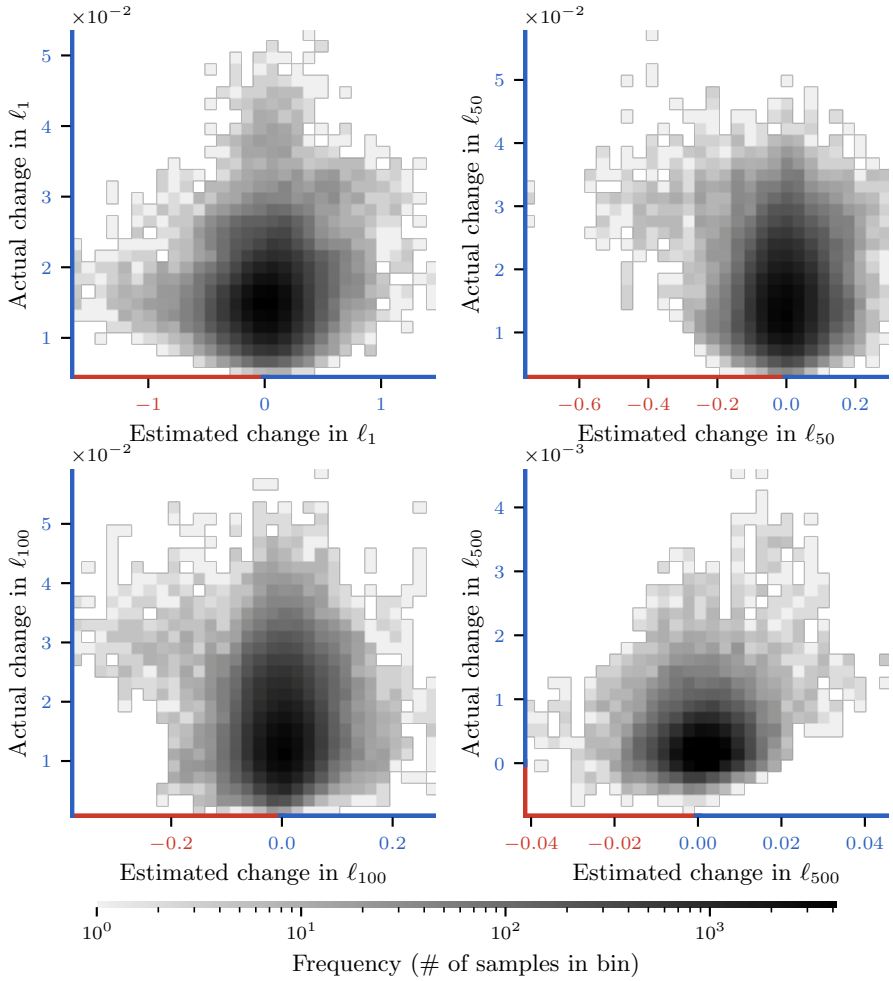


Figure 14: Change in per-diffusion-timestep losses ℓ_t when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for the `CIFAR-2` dataset, for measurements $\ell_t(x, \theta)$ 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 ($\ell_5 00$), this pattern seems to subside. Losses ℓ_t in the $200 - 500$ range seem to work well for predicting changes in other losses Figure 9.

results might suggest that models trained on different subsets of data eventually start behaving the same if the number of data points is sufficiently large, but Figures 15 and 16 would imply that the thresholds of sufficient data size might differ at different diffusion timesteps.

J Retraining without top influences

The counterfactual retraining results are shown in Figure 18 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.

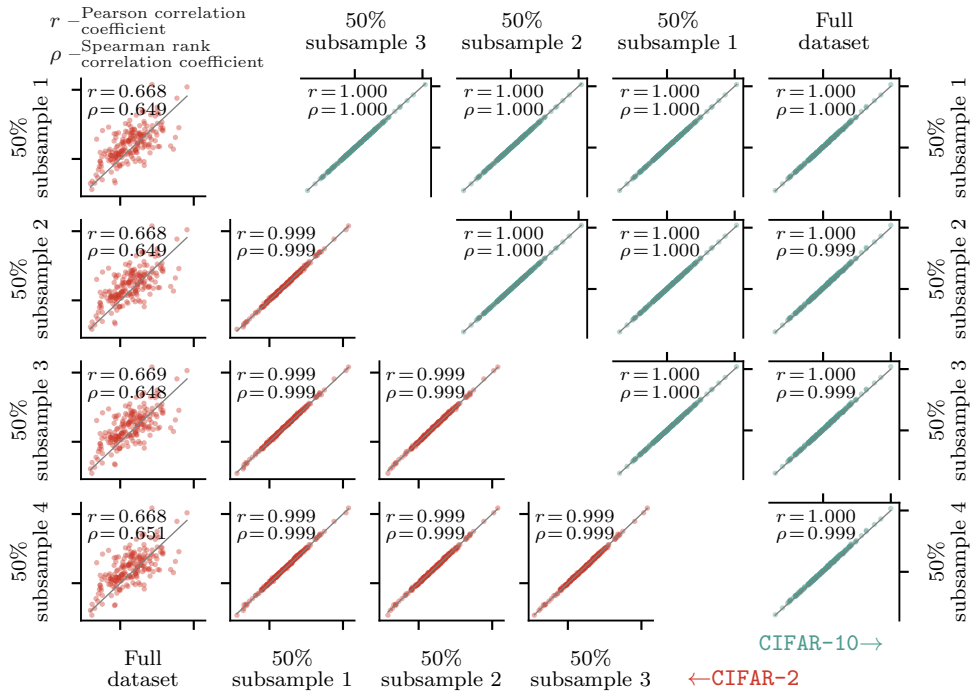


Figure 15: Correlation of the $\mathrm{ELBO}(x, \theta)$ measurements on different data points x (samples generated from the model trained on full data), for models trained on different subsets of data. Each subplot plots $\mathrm{ELBO}(x, \theta)$ measurements for 200 generated samples x, as measured 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.

K Hessian Approximation Ablation

In Figure 19, 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 (Appendix D.1). We find that the better-motivated "MC-Fisher" estimator $\mathrm{GGN}^{\texttt{model}}$ in Equation (7) does indeed perform better than the "empirical Fisher" in Equation (9) 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 [6]. 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 Eschenhagen et al. [6]. 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.

L LDS results for probability of sampling trajectory

The results for the "log probability of sampling trajectory" measurements are shown in Figure 20. 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

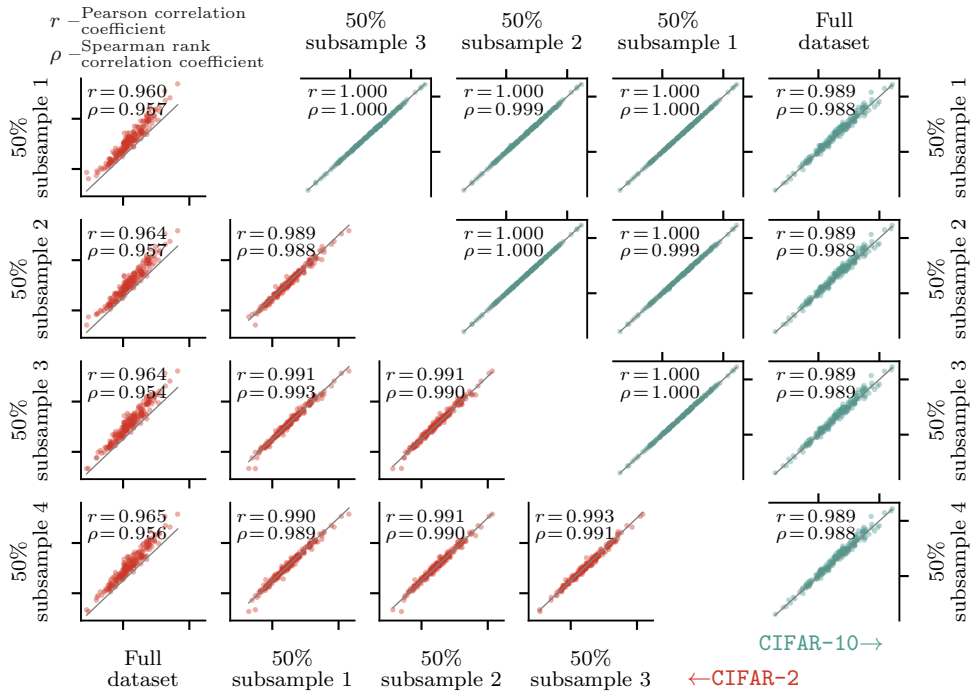


Figure 16: Correlation of the diffusion loss $\ell(x, \theta)$ measurements on different data points x (samples generated from the model trained on full data), for models trained on different subsets of data. See the caption of Figure 15 for details, the plot is identical except for the measurement being the diffusion loss rather than ELBO.

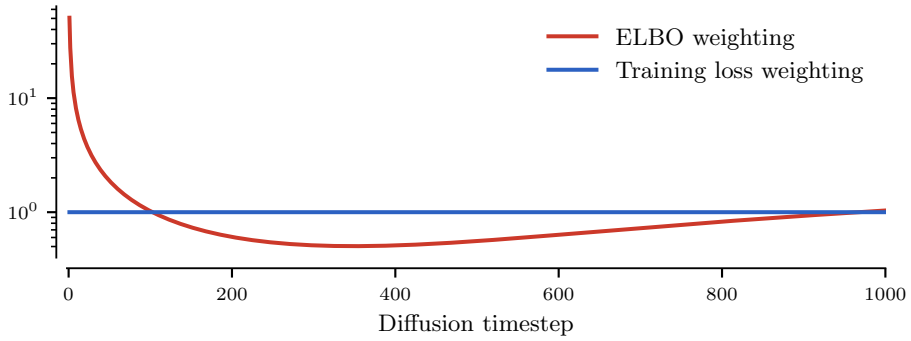


Figure 17: The diffusion loss and diffusion ELBO as formulated in [11] (ignoring the reconstruction term that accounts for the quantisation of images back to pixel space) are equal up to the weighting of the individual per-diffusion-timestep loss terms and a constant independent of the parameters. This plot illustrates the relatives difference in the weighting for per-diffusion-timestep losses applied in the ELBO vs. in the training loss.

unsurprising, since the measurement comprises the log-densities of particular values of 1000 latent variables.

M Experimental details

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

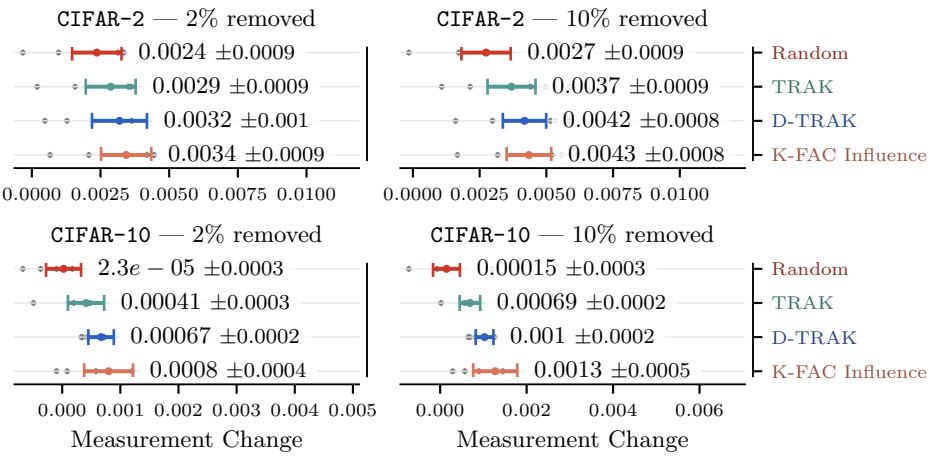


Figure 18: Changes in measurements under counterfactual retraining without top influences for the **loss** measurement. The standard error in the estimate of the mean is indicated with error bars and reported after '±', where the average is over different generated samples for which top influences are being identified.

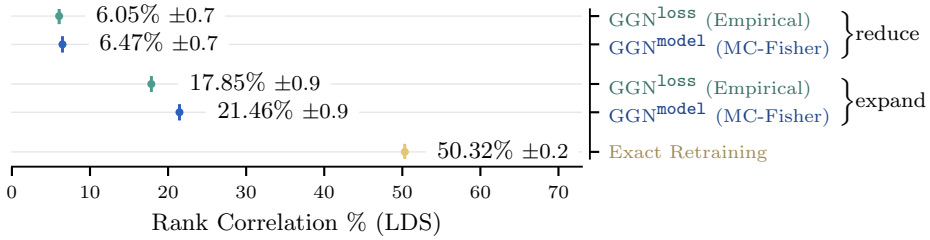


Figure 19: Ablation over the different Hessian approximation variants introduced in Section 3.1. We ablate two versions of the GGN: the "MC" Fisher in Equation (7) and the "Empirical" Fisher in Equation (9), as well as two settings for the K-FAC approximation: "expand" and "reduce".

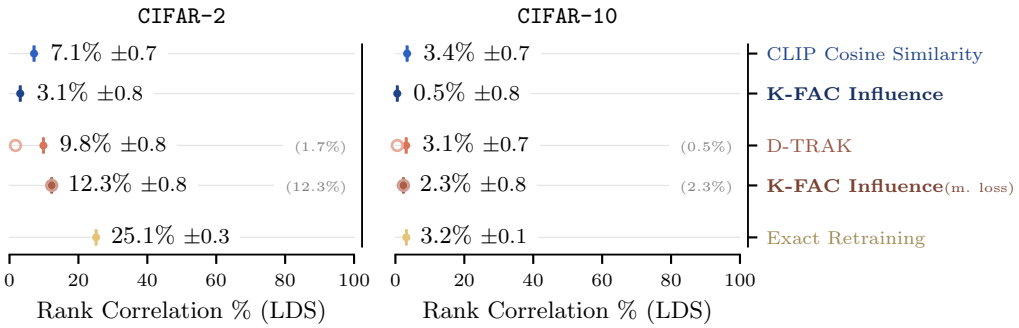


Figure 20: Linear Data-modelling Score (LDS) for the **probability of sampling trajectory**. The plot follows the same format as that of Figures 2a and 2b. Overall, probability of the sampling trajectory appears to be a difficult proxy for the marginal probability of sampling a given example, given that 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).

M.1 Data attribution baselines

In our framework, their method can be tersely described as using $\mathrm{GGN}^{\mathtt{loss}}_{\mathcal{D}}$ (Empirical Fisher) in Equation (9) as a Hessian approximation instead of $\mathrm{GGN}^{\mathtt{model}}_{\mathcal{D}}$ (MC-Fisher) in Equation (8), and computing the Hessian-preconditioned inner products using random projections [5] 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 Zheng et al. [30] 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 [22] 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.

M.2 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 [16]. We use 50000 images (the train split) for training.

`CIFAR-2` For `CIFAR-2`, we follow Zheng et al. [30] 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.

M.3 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 [11]. This U-Net architecture contains both convolutional and attention layers. We use the same noise schedule as described for the `CIFAR` dataset in [11].

Sampling We follow the standard DDPM sampling procedure with a full 1000 timesteps to create the generated samples as described by Ho et al. [11]. DDPM sampling usually gives better samples (in terms of visual fidelity) than Denoising Diffusion Implicit Models (DDIM) sampling [28] 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 $\mathcal{N}\big(x^{(t-1)}|\mu_{t-1|t,0}\big(x^{(t)}, \epsilon_\theta^t(x^{(t)})\big), \sigma_t^2 I\big)$ 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 Ho et al. [11, §3.2], i.e. we set:

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

M.4 Details on data attribution methods

TRAK For TRAK baselines, we adapt the implementation of Park et al. [21], Georgiev et al. [7] to the diffusion modelling setting. When running TRAK, there are several settings the authors recommend to consider: **1)** the projection dimension $d_{\texttt{proj}}$ 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 [30]. 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 Figure 21, 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 6 and 8, 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 [22]. For this baseline, we compute the CLIP embeddings [22] 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 [21] 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 19, 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 (4)), includes multiple nested expectations over diffusion timestep $\tilde{t}$ and noise added to the data $\epsilon^{(t)}$. This is also the case for the $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ in Equation (7) and for the gradients $\nabla_\theta \ell(\theta, x_n)$ in the computation of $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ in Equation (9), 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 $\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$, we explicitly iterate over all training data points, and draw K samples of $\left(\tilde{t}, \epsilon^{(\tilde{t})}, x_n^{(\tilde{t})}\right)$ for each datapoint. For $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$, we explicitly iterate over all training data points, and draw K samples of $\left(\tilde{t}, \epsilon^{(\tilde{t})}, x_n^{(\tilde{t})}\right)$ to compute the gradients $\nabla_\theta \ell(\theta, x_n)$ before taking an outer product. Note that, for $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$, 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.

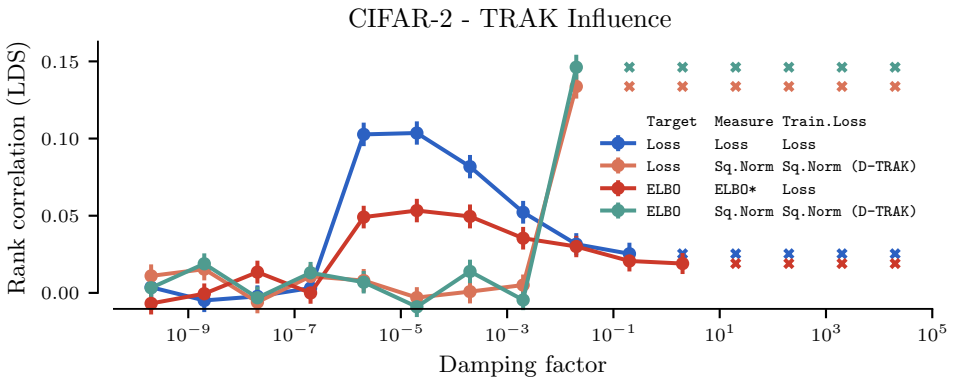


Figure 21: LDS scores on for TRAK (random projection) based influence on `CIFAR-2` when using half-precision (`float16`) for influence computations. Compare with Figure 6. `NaN` results are indicated with $\times$.

M.5 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 $H \approx \nabla_\theta^2 \mathcal{L}_{\mathcal{D}} = \nabla_\theta^2 1/N \sum \ell(\theta, x_n)$, we add a damping factor λ to

the diagonal before inversion:

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

where I is a $d_{\texttt{param}} \times d_{\texttt{param}}$ 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 (9), 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 [30]). In other words, if $P \in \mathbb{R}^{d_{\texttt{proj}} \times d_{\texttt{param}}}$ is the projection matrix (see [21] for details), then damped Hessian-inverse preconditioned vector inner products between two vectors $v_1, v_2 \in \mathbb{R}^{d_{\texttt{param}}}$ (e.g. the gradients in Equation (4)) would be computed as:

$$(Pv_1)^\intercal (H + \lambda I)^{-1} Pv_,.$$

where $H \approx P \nabla^2_\theta \mathcal{L}_\mathcal{D} P^\intercal \in \mathbb{R}^{d_{\texttt{proj}} \times d_{\texttt{proj}}}$ 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 [3]. For TRAK-based methods, Park et al. [21] recommend using no damping: "[...] computing TRAK does not require the use of additional regularization (beyond the one implicitly induced by our use of random projections)" [21, § 6]. 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 [30] are larger by a factor of N (the dataset size) than the equivalent damping values reported in this paper.

M.6 LDS Benchmarks

For all LDS benchmarks [21], we sample 100 sub-sampled datasets ($M := 100$ in Equation (17)), and we train 5 models with different random seeds ($K := 5$ in Equation (17)), 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.

M.7 Retraining without top influences

For the retraining without top influences experiments (Figure 18), 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.

M.8 Training details

For `CIFAR-10` and `CIFAR-2` we again follow the training procedure outlined in [11], with the only difference being a shortened number of training iterations. For `CIFAR-10`, we train for 160000 steps (compared to 800000 in [11]) 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 [30], 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.

M.9 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 (1) 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 Section 2.2.1), **2)** when computing the per-example train loss gradients for influence functions, **3)** when computing the loss measurement function.

When computing $\mathrm{GGN}_\mathcal{D}^{\texttt{model}}$ in Equation (7), 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 $\mathbb{E}_{x_n}\mathbb{E}_{\tilde{x}_n}$:

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

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:

$$\mathrm{F}_\mathcal{D}(\theta) = \mathbb{E}_{x_n}\left[\mathbb{E}_{\tilde{x}_n}\left[\mathbb{E}_{\tilde{t}}\left[\mathbb{E}_{x_n^{(\tilde{t})},\epsilon^{(\tilde{t})}}\mathbb{E}_{\epsilon_{\texttt{mod}}}\left[g_n(\theta)g_n(\theta)^\intercal\right]\right]\right]\right], \qquad \epsilon_{\texttt{mod}} \sim \mathcal{N}\left(\epsilon_\theta^{\tilde{t}}\left(x_n^{(\tilde{t})}\right), I\right),$$

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

When computing $\mathrm{GGN}_\mathcal{D}^{\texttt{loss}}$ in Equation (9), 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 [21] in prior works that rely on an empirical GGN [30, 7].[5] Hence, the GGN in Equation (9) takes the form:

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

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 $\mathrm{GGN}_\mathcal{D}^{\texttt{model}}$ 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 Equation (3), we again simply replace the per-example training loss $\ell(\theta^\star, x_j)$ with the per-example training loss averaged over data augmentations $\tilde{\ell}(\theta^\star, x_j)$, 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 (4), 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 $\tilde{t}$ and noise samples $\epsilon^{(t)}$), we simply report the total number of MC samples per data point, where data augmentations, diffusion time $\tilde{t}$, etc. are all drawn independently for each sample.

[5] 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.