INFLUENCE FUNCTIONS FOR SCALABLE DATA ATTRIBUTION IN DIFFUSION MODELS

Bruno Mlodozeniec
University of Cambridge, MPI Tübingen

Runa Eschenhagen
University of Cambridge

Juhan Bae
Anthropic

Alexander Immer
ETH Zurich, MPI Tübingen

David S. Krueger

Richard Turner
University of Cambridge

ABSTRACT

Diffusion models have led to significant advancements in generative modelling. Yet their widespread adoption poses challenges regarding data attribution and interpretability. In this paper, we aim to help address such challenges in diffusion models by extending *influence functions*. 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 use a K-FAC approximation based on generalised Gauss-Newton matrices specifically tailored to diffusion models. We show that our recommended method outperforms previously proposed data attribution methods on common data attribution evaluations, such as the Linear Data-modelling Score (LDS) or retraining without top influences, without the need for method-specific hyperparameter tuning.

1 INTRODUCTION

Generative modelling for continuous data like images, video and audio, has advanced rapidly driven 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 (Schuhmann et al., 2022), concerns about the extent to which individual copyright owners' works influence the generated samples arise. Some already characterise what companies offer as "copyright infringement as a service" (Saveri & Butterick, a), which has caused a flurry of high-profile lawsuits Saveri & Butterick (a;b). 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 (Koh & Liang, 2017; Bae et al., 2022) offer precisely such a tool. By approximating the answer to the question, "If the model was not trained on some of the data, what would its output be?", they can help finding data points most responsible for a low loss on an example, or high probability of generating a particular example. However, they have yet to be scalably adapted to the general diffusion modelling setting.

Influence functions work by locally approximating how the loss landscape would change if some of the training data points were down-weighted in the training loss (illustrated in Figure 5). Consequently, this enables prediction for how the (local) optimum of the training loss would change, and how that change in the parameters would affect a measurement of interest (e.g., loss on a particular example). By extrapolating this prediction, one can make an estimate for what would happen if the data points were fully removed from the training set. However, to locally approximate the shape of the loss

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.

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, Schraudolph, 2002; Martens, 2020). The GGN has not been 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. A popular approach to approximating the GGN is Kronecker-Factored Approximate Curvature (K-FAC, Heskes, 2000; Martens & Grosse, 2015). It is not commonly known how to apply it to neural network architectures used in diffusion models; for example, Kwon et al. (2023) resort to alternative Hessian approximation methods because "[K-FAC] might not be applicable to general deep neural network models as it highly depends on the model architecture". However, based on previous work it is indeed clear that it can be applied to architectures used in diffusion models (Grosse & Martens, 2016; Eschenhagen et al., 2023), which typically combine linear layers, convolutions, and attention Ho et al. (2020).

In this work, we describe an approach to scalable influence function-based approximations for data attribution in diffusion models, using a K-FAC approximation of a GGN as the Hessian approximation. We articulate a design space based on influence functions, unify previous methods for data attribution in diffusion models (Georgiev et al., 2023; Zheng et al., 2024) 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 (Park et al., 2023) or retraining without top influences. Finally, we also discuss interesting empirical observations that challenge our current understanding of influence functions in the context of diffusion models.

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 $\theta \in \mathbb{R}^{d_{\text{param}}}$ to approximate a training data distribution $q(x)$, with the primary aim being to sample new data $x \sim p_\theta(\cdot)$ (Sohl-Dickstein et al., 2015; Ho et al., 2020; Turner et al., 2024). This

is usually done by augmenting the original data x with T fidelity levels as $x^{(0:T)} = [x^{(0)}, \ldots, x^{(T)}]$ with an augmentation distribution $q(x^{(0:T)})$ that satisfies the following criteria: **1)** the highest fidelity $x^{(0)}$ equals the original training data $q(x^{(0)}) = q(x)$, **2)** the lowest fidelity $x^{(T)}$ has a distribution that is easy to sample from, and **3)** predicting a lower fidelity level from the level directly above it is simple to model and learn. To achieve the above goals, q is typically taken to be a first-order Gaussian auto-regressive (diffusion) process: $q(x^{(t)}|x^{(0:t-1)}) = \mathcal{N}(x^{(t)}|\lambda_t x^{(t-1)}, (1-\lambda_t)^2 I)$, with hyperparameters λ_t set so that the law of $x^{(T)}$ approximately matches a standard Gaussian distribution $\mathcal{N}(0, I)$. In that case, the reverse conditionals $q(x^{(t-1)}|x^{(t:T)}) = q(x^{(t-1)}|x^{(t)})$ are first-order Markov, and if the number of fidelity levels T is high enough, they can be well approximated by a diagonal Gaussian, allowing them to be modelled with a parametric model with a simple likelihood function, hence satisfying **(3)** (Turner et al., 2024). 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:

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

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

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

as in Equation (1). In other words, the mean is a mixture of the sample $x^{(t)}$ and the noise $\epsilon^{(t)}$ that was applied to $x^{(0)}$ to produce it. Hence, we can choose to analogously parameterise $p_\theta(x^{(t-1)}|x^{(t)})$ as $\mathcal{N}\left(x^{(t-1)}|\mu_{t-1|t,0}\left(x^{(t)}, \epsilon_\theta^t(x^{(t)})\right), \sigma_t^2 I\right)$. 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 hyper-parameters (Ho et al., 2020). 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 (Ho et al., 2020; Turner et al., 2024):[1]

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

This leads to a training loss ℓ for the diffusion model $\epsilon_\theta^t(x^{(t)})$ that is a sum of per-diffusion time-step 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 $\mathcal{D}$=$\{x_n\}_{n=1}^N$:

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

Other interpretations of the above procedure exist in the literature (Song & Ermon, 2020; Song et al., 2021b;a; Kingma et al., 2023).

2.2 Influence Functions

The aim of influence functions is to answer questions of the sort "how would my model behave were it trained on the training dataset with some datapoints removed". To do so, they approximate the change in the optimal model parameters in Equation (3) when some training examples $(x_{i_1}, \ldots, x_{i_M})$ are removed from the dataset $\mathcal{D}$. To arrive at a tractable approximations, it is useful to consider a

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

continuous relaxation of this question: how would the optimum change were the training examples $(x_{i_1}, \ldots, x_{i_M})$ down-weighted by $\varepsilon \in \mathbb{R}$ in the loss function:

$$r_{-(i_1,\ldots,i_M)}(\varepsilon) = \arg\min_{\theta} \frac{1}{N} \sum_{n=1}^{N} \ell(\theta, x_n) - \varepsilon \sum_{j=1}^{M} \ell(\theta, x_{i_j}) \tag{4}$$

The function $r_{-(i_1,\ldots,i_M)} : \mathbb{R} \to \mathbb{R}^{d_{\text{param}}}$ (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 (3) with examples $(x_{i_1}, \ldots, x_{i_M})$ removed.

Under suitable assumptions (see Appendix A), by the Implicit Function Theorem (Krantz & Parks, 2003), the response function is continuous and differentiable at $\varepsilon = 0$. *Influence functions* can be defined as a linear approximation to the response function $r_{-(i_1,\ldots,i_M)}$ by a first-order Taylor expansion around $\varepsilon = 0$:

$$\begin{aligned} r_{-(i_1,\ldots,i_M)}(\varepsilon) &= r_{-(i_1,\ldots,i_M)}(0) &&+ \left.\frac{dr_{-(i_1,\ldots,i_M)}(\varepsilon')}{d\varepsilon'}\right|_{\varepsilon'=0} \varepsilon &&+ o(\varepsilon) \\ &= \theta^\star(\mathcal{D}) &&+ \sum_{j=1}^{M} \left(\nabla_\theta^2 \mathcal{L}_\mathcal{D}(\theta)\right)^{-1} \nabla_\theta \ell(\theta, x_{i_j}) \varepsilon &&+ o(\varepsilon), \end{aligned} \tag{5}$$

as $\varepsilon \to 0$. See Appendix A for a formal derivation and conditions. Again, the optimal parameters with examples $(x_{i_1}, \ldots, x_{i_M})$ fully removed can be approximated by setting ε to $1/N$.

Usually, we are not directly interested in the change in parameters in response to removing some data, but rather the response 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 $\theta^\star(\mathcal{D})$ — $m(\theta, x') = m(\theta^\star, x') + \nabla_{\theta^\star}^\intercal m(\theta^\star, x')(\theta - \theta^\star) + o\left(\|\theta - \theta^\star\|_2\right)$ — and combine it with influence functions to get a simple linear estimate of the change in the measurement function:

$$m(r_{-(i_1,\ldots,i_M)}(\varepsilon), x') = m(\theta^\star, x') + \sum_{j=1}^{M} \nabla_{\theta^\star}^\intercal m(\theta^\star, x') \left(\nabla_{\theta^\star}^2 \mathcal{L}_\mathcal{D}(\theta^\star)\right)^{-1} \nabla_{\theta^\star} \ell(\theta^\star, x_{i_j}) \varepsilon + o(\varepsilon). \tag{6}$$

2.2.1 Generalised Gauss-Newton matrix

Computing the influence function approximation in Equation (5) requires inverting the Hessian $\nabla_\theta^2 \mathcal{L}_\mathcal{D}(\theta) \in \mathbb{R}^{d_{\text{param}} \times d_{\text{param}}}$. In the context of neural networks, the Hessian itself is generally computationally intractable and approximations are necessary. A common Hessian approximation is the positive semidefinite generalised Gauss-Newton matrix (GGN).

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

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

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 $\mathbb{E}_z[\nabla_\theta^2 \rho(\theta, z)]$. 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 (Martens, 2020).

Option 1. Typically, f_z would be the neural network function on a training datapoint z, and h_z would 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 $\text{GGN}^{\texttt{model}}$. The GGN with this split is exact for linear neural networks (or when the model has zero residuals on the training data) (Martens, 2020).

$$\begin{aligned} f_z &\coloneqq \text{mapping from parameters to model output} \\ h_z &\coloneqq \text{loss function (e.g. } \ell_2\text{-loss)} \end{aligned} \quad \to \text{GGN}^{\texttt{model}}(\theta) \tag{7}$$

Option 2. Alternatively, a different GGN can be defined by using a trivial split of the loss $\rho(\theta, z)$ into the identity map $h_z \coloneqq \text{id}$ 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

$$\begin{aligned} f_z &\coloneqq \rho(\cdot, z) \\ h_z &\coloneqq \mathrm{id} \end{aligned} \qquad \to \mathrm{GGN}^{\texttt{loss}}(\theta) = \mathbb{E}_z \left[\nabla_\theta \rho(\theta, z) \nabla_\theta^\intercal \rho(\theta, z)\right]. \tag{8}$$

This is also called the empirical Fisher (Kunstner et al., 2019). Note that $\mathrm{GGN}^{\texttt{loss}}$ is only equal to the Hessian under the 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 generally prefer to use $\mathrm{GGN}^{\texttt{model}}$ whenever we have a nonlinear loss, and we're interested in approximating the Hessian.

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 (Georgiev et al., 2023; Zheng et al., 2024) 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.

3.1.1 GGN for diffusion models

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

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

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

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

where $g_n(\theta) = \nabla_\theta \|\epsilon_{\mathrm{mod}}^{(\tilde{t})} - \epsilon_\theta^{\tilde{t}}(x_n^{(\tilde{t})})\|^2$; see Appendix B for the derivation. This formulation lends itself to a Monte Carlo approximation, since we can now compute gradients using auxiliary targets $\epsilon_{\mathrm{mod}}^{(\tilde{t})}$ sampled from the model's output distribution, as shown in Equation (10). $\mathrm{F}_{\mathcal{D}}$ can be interpreted as a kind of Fisher information matrix (Amari, 1998; Martens, 2020), but it is not the Fisher for the marginal model distribution.

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

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

where $\ell(\theta, x)$ is the diffusion training loss defined in Equation (2). 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 (Georgiev et al., 2023; Zheng et al., 2024). In contrast, in this work, we opt for

$\mathrm{GGN}_{\mathcal{D}}^{\texttt{model}}$ in Equation (9), 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. (2024), 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 (9).[2] Hence, in our method substituting different targets would not affect the Hessian approximation. In Zheng et al. (2024) replacing the targets only makes a difference to the Hessian approximation because they chose $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ (an empirical Fisher) to approximate the Hessian.

3.1.2 K-FAC FOR DIFFUSION MODELS

While $\mathrm{F}_{\mathcal{D}}(\theta)$ and $\mathrm{GGN}_{\mathcal{D}}^{\texttt{loss}}$ do not require computing full Jacobians or the Hessian of the neural network model, they involve taking outer products of gradients of size $\mathbb{R}^{d_{\texttt{param}}}$, which is still intractable. Kronecker-Factored Approximate Curvature (Heskes, 2000; Martens & Grosse, 2015, 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 (6) efficient enough to scale to large networks. K-FAC is defined for linear layers, including linear layers with weight sharing like convolutions (Grosse & Martens, 2016); when weight sharing is used, there are two variations – K-FAC-expand and K-FAC-reduce (Eschenhagen et al., 2023). This covers most layer types in the architectures typically used for diffusion models. Therefore, we choose to approximate the Hessian with a K-FAC approximation of $\mathrm{F}_{\mathcal{D}}$, similar to Grosse et al. (2023).

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

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

with $a_n^{(l)} \in \mathbb{R}^{d_{\text{in}}^l}$ being the inputs to the lth layer for data point $x_n^{(\tilde{t})}$ and $b_n^{(l)} \in \mathbb{R}^{d_{\text{out}}^l}$ being the gradient of the ℓ_2-loss w.r.t. the output of the lth layer.[3] The approximation trivially becomes an equality for a single data point and also for deep linear networks with ℓ_2-loss (Bernacchia et al., 2018; Eschenhagen et al., 2023). We approximate the expectations in Equation (12) with Monte Carlo samples and use K-FAC-expand whenever weight sharing is used since the problem formulation of diffusion models corresponds to the expand setting in Eschenhagen et al. (2023); in the case of convolutional layers this corresponds to Grosse & Martens (2016). Lastly, to ensure the Hessian approximation is well-conditioned and invertible, it is standard practice to add a small damping term to the diagonal entries. We ablate these design choices in Section 4 (Figures 4, 7 and 9).

3.2 GRADIENT COMPRESSION AND QUERY BATCHING

In practice, we recommend computing influence function estimates in Equation (6) by first computing and storing the approximate Hessian inverse, and then iteratively computing the preconditioned inner products $\nabla_{\theta^\star}^\intercal m(\theta^\star, x) \left(\nabla_{\theta^\star}^2 \mathcal{L}_{\mathcal{D}}(\theta^\star)\right)^{-1} \nabla_{\theta^\star} \ell(\theta^\star, x_j)$ for different training datapoints x_j. Following Grosse et al. (2023), 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 compared to the SVD-based compression used by (Grosse et al., 2023) (see Appendix C), likely due to the fact that for diffusion models gradients are not low-rank.

3.3 WHAT TO MEASURE

For diffusion models, arguably the most natural question to ask might be, for a given sample x generated from the model, how did the training samples influence the probability of generating a sample x? For example, in the context of copyright infringement, we might want to ask if removing

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

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

certain copyrighted works would substantially reduce the probability of generating x. With influence functions, these questions could be interpreted as setting the measurement function $m(\theta, x)$ to be the (marginal) log-probability of generating x from the diffusion model: $\log p_\theta(x)$.

Computing the marginal log-probability introduces some challenges. Diffusion models have originally been designed with the goal of tractable sampling, and not log-likelihood evaluation; Ho et al. (2020); Sohl-Dickstein et al. (2015) only introduce a lower-bound on the marginal log-probability. Song et al. (2021b) show that exact log-likelihood evaluation is possible, but it only makes sense in settings where the training data distribution has a density (e.g. uniformly dequantised data), and it only corresponds to the marginal log-likelihood of the model when sampling deterministically (Song et al., 2021a).[4] Also, taking gradients of that measurement, as required for influence functions, is non-trivial. Hence, in most cases, we might need a proxy measurement for the marginal probability. We consider a couple of approaches to approximately measuring log-likelihood in this work:

1. **Loss** Approximate $\log p_\theta(x)$ with the diffusion loss $\ell(\theta, x)$ in Equation (2) on that particular example. This corresponds to the ELBO with reweighted per-timestep loss terms (see Figure 16 in the appendix).
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 (ELBO) (Ho et al., 2020).

4 Experiments

Evaluating Data Attribution. To evaluate the proposed data attribution methods, we primarily focus on two metrics: *Linear Data Modelling Score* (LDS) and *retraining without top influences*. LDS measures how well a given attribution method can predict the relative magnitude in the change in a measurement as the model is retrained on (random) subsets of the training data. For an attribution method $a(\mathcal{D}, \mathcal{D}', z)$ that approximates how a measurement $m(\theta^*(\mathcal{D}'), z)$ 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:

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

where $\tilde{\mathcal{D}}_i$ are independently subsampled versions of the original dataset $\mathcal{D}$, each containing 50% of the points sampled without replacement. Since, in deep learning, training on a fixed dataset can produce different models with different measurements depending on the random seed, Park et al. (2023) propose to use an ensemble average measurement after retraining as the "oracle" target:

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

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

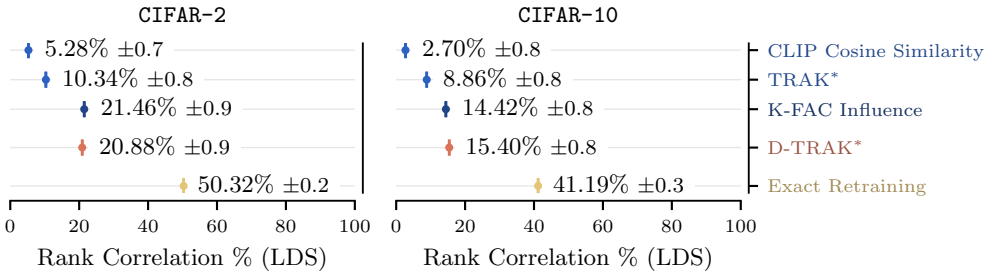
Retraining without top influences, on the other hand, evaluates the ability of the data attribution method to surface the most influential datapoints – namely, those that would most negatively affect the measurement $m(\theta^*(\mathcal{D}'), z)$ under exact counterfactual retraining when removed from the training set $\mathcal{D}'$. For each method, we remove a fixed percentage of the most influential datapoints from $\mathcal{D}$ to create the new dataset $\mathcal{D}'$, and report the change in the measurement $m(\theta^*(\mathcal{D}'), z)$ relative to $m(\theta^*(\mathcal{D}), z)$ measured by the model traind on the full dataset $\mathcal{D}$, averaging over different examples z.

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) (Ho et al., 2020) throughout.

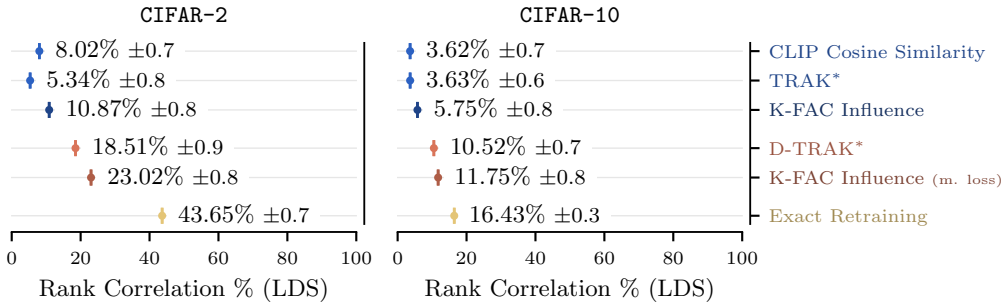
[4] Unless the trained model satisfies very specific "consistency" constraints (Song et al., 2021b, Theorem 2).

[4] Better LDS results can sometimes be obtained when looking at validation examples (Zheng et al., 2024), but diffusion models are used primarily for sampling, so attributing generated is of primary practical interest.

We compare influence functions with K-FAC (K-FAC Influence) to TRAK as formulated for diffusion models in Georgiev et al. (2023); Zheng et al. (2024). In our framework, their method can be tersely described as using the Empirical Fisher (Equation (11)) as a Hessian approximation, and computing the Hessian-preconditioned inner products using random projections Dasgupta & Gupta (2003) 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. (2024) to give improved performance on LDS benchmarks.



(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. For methods marked with $*$, best results across a hyperparameter (damping factor) sweep are reported. Methods that substitute in *incorrect* measurement functions into the approximation are separated and plotted with $\bullet$. "(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 $\bullet$. Standard error in the LDS score estimate is indicated with '$\pm$'.

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, and sometimes outperforms the hand-tuned changes in D-TRAK. Furthermore, TRAK and D-TRAK appear to be significantly more sensitive to tuning the damping factor. These methods appear to not 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/method (Appendix D). In Figures 2a and 2b, we report the results for TRAK and D-TRAK for the best damping values from a sweep (see Appendix D). In contrast, for K-FAC Influence, we default to a small damping value for numerical stability of 10^{-8} throughout, as done in prior work (Bae et al., 2024). We find that generally any sufficiently small value works reasonably well if enough samples are taken for estimating the loss and measurement (Figures 7 and 9). When tuning the damping factor, KFAC Influence can achieve $18.3\%_{\pm 0.7}$ LDS score for `CIFAR-10` in Figure 2a, and hence consistently outperforms TRAK and D-TRAK with the same amount of tuning. 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.

Retraining without top influences. The counterfactual retraining results are shown in Figure 3 for CIFAR-2, CIFAR-10, with 2% and 10% of the data removed. In this evaluation, influence functions with K-FAC consistently pick more influential training examples (i.e. those which lead to a higher loss reduction) than the baselines.

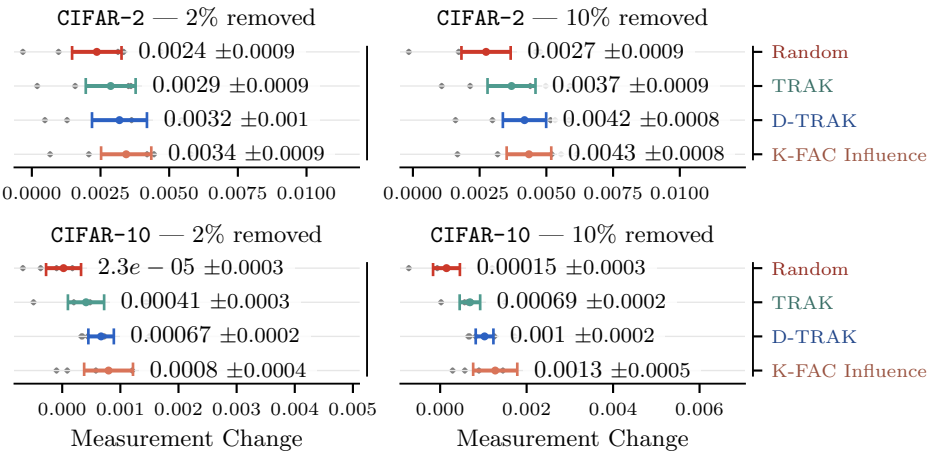


Figure 3: Changes in measurements under counterfactual retraining without top influences for the **loss** measurement.

Hessian Approximation Ablation. In Figure 4, we explore the impact of the Hessian approximation design choices discussed in Section 3.1. We use K-FAC to approximate the GGN in all cases, with either the "expand" or the "reduce" variant (Section 3.1.2). We find that the better-motivated "MC-Fisher" approximation to $\text{GGN}^{\texttt{model}}$ in Equation (9) does indeed perform better than the "empirical Fisher" in Equation (11) used in TRAK and D-TRAK. Secondly, we find that K-FAC expand significantly outperforms K-FAC reduce, which stands in contrast to the results in the second-order optimisation setting where the two are on par with one another (Eschenhagen et al., 2023). 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. (2023). Hence, our results all seem to imply that a better Hessian approximation directly results in better downstream data attribution performance. However, we do not directly evaluate the approximation quality of the estimates and also do not sweep over the damping value for all variants.

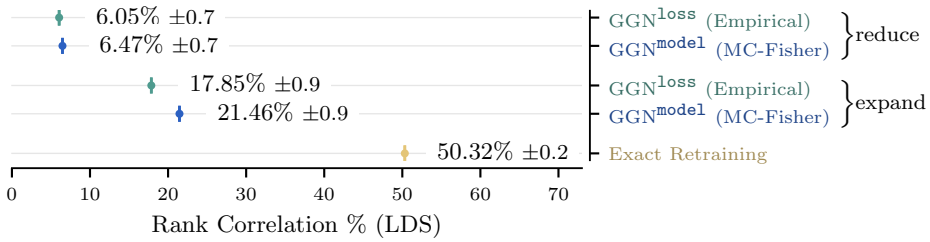


Figure 4: Ablation over the different Hessian approximation variants introduced in Section 3.1. We ablate two versions of the GGN: the "MC" Fisher in Equation (9) and the "Empirical" Fisher in Equation (11), as well as two settings for the K-FAC approximation: "expand" and "reduce" (Eschenhagen et al., 2023).

4.1 Potential challenges to use of influence functions for diffusion models

One peculiarity in the LDS results, similar to the findings in Zheng et al. (2024), 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)").[5] As illustrated in Figure 16, 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

[5]Note that, unlike Zheng et al. (2024), we only change the measurement function for a proxy in the influence function approximation, keeping the Hessian approximation and training loss gradient in Equation (6) the same.

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 for different substituting *different* per-diffusion-timestep losses in the K-FAC influence approximation. The results are shown in Figure 11, leading to the following observation:

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

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

In Figure 12, we show the same LDS scores, but with the influence "estimated" by actually retraining a model, rather than using an influence function approximation. Here, substituting the wrong measurement $\ell_{t'}$ for predicting the change in ℓ_t can yield a decent estimate if t and t' are close. Unsurprisingly, there is no same pattern where using $t' > t$ yields a significantly better prediction than using the correct timestep for the estimating measurement. Hence, the peculiar behaviour in Figure 11 is specific to the approximations used in the influence functions.

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 mostly increases loss.*

We show in Figures 13 and 14 that influence functions tend to overestimate how often removal of a data point will lead to improvements in loss on a generated sample (both for aggregate diffusion training loss in Section 2.1, and the per-diffusion-timestep loss in Equation (2)).

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

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

As illustrated in Figure 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 if one sample is more likely to be generated than another sample by one model (as measured by ELBO), it will be less likely to be generated than another by a model trained on a different random subset of the data. This implies that the standard DDPM ELBO might be a poor proxy for the real question we intend to ask: "how likely is a given sample to be generated by a model?". It would appear improbable that the true probabilities of generating any given sample are close-to unaffected by resampling what data subset the model was trained on.

5 Discussion

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

This appears to be both due to the limitations of the influence functions approximation, but also due to the shortcomings of the considered proxy measurements (Section 4.1).

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

REFERENCES

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

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

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

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

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.

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

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

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

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.

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

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.

Diederik P. Kingma, Tim Salimans, Ben Poole, and Jonathan Ho. Variational Diffusion Models, April 2023.

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.

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.

Alex Krizhevsky. Learning Multiple Layers of Features from Tiny Images. 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

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

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.

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

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

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

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

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

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

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.

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

Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution, 2020. URL https://arxiv.org/abs/1907.05600.

Yang Song, Conor Durkan, Iain Murray, and Stefano Ermon. Maximum Likelihood Training of Score-Based Diffusion Models, October 2021a.

Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-Based Generative Modeling through Stochastic Differential Equations, February 2021b.

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.

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

A Derivation of Influence Functions

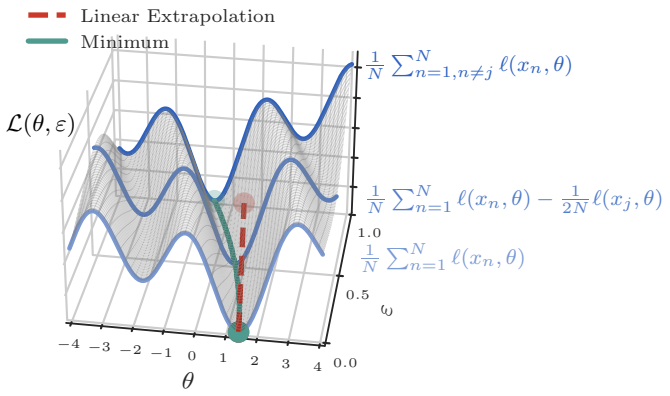


Figure 5: Illustration of the influence function approximation. Influence funcitons consider the extended loss landscape $\mathcal{L}(\theta, \varepsilon) \stackrel{\text{def}}{=} \frac{1}{N}\sum_{n=1}^{N} \ell(x_n, \theta) - \frac{\varepsilon}{N}\ell(x_j, \theta)$, where the loss $\ell(x_j, \theta)$ 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 (●).

A.1 Implicit Function Theorem

Theorem 1 (Implicit Function Theorem (Krantz & Parks, 2003)) *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 $J_{F,\mathbf{y}}(\mathbf{a}, \mathbf{b}) \in \mathbb{R}^{m \times m}$ of $\mathbf{y} \mapsto F(\mathbf{a}, \mathbf{y})$:*

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

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:*

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

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:

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

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:

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

Rearranging gives equation Equation (14).

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

Consider a loss function $\mathcal{L} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ that depends on some hyperparameter $\boldsymbol{\epsilon} \in \mathbb{R}^n$ (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 $\mathcal{L}(\boldsymbol{\epsilon}, \boldsymbol{\theta})$, the derivative with respect to the parameters $\boldsymbol{\theta}$ will be zero. Hence, assuming that the loss function is twice continuously differentiable (hence $\frac{\partial L}{\partial \boldsymbol{\epsilon}}$ is continuously differentiable), and assuming that for some $\boldsymbol{\epsilon}' \in \mathbb{R}^n$ we have a set of parameters $\boldsymbol{\theta}^\star$ such that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\epsilon}}(\boldsymbol{\epsilon}', \boldsymbol{\theta}^\star) = \mathbf{0}$ and the Hessian $\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{\theta}^2}(\boldsymbol{\epsilon}', \boldsymbol{\theta}^\star)$ is invertible, we can apply the implicit function theorem to the derivative of the loss function $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\epsilon}} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^m$, to get the existence of a continuously differentiable function g such that $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\epsilon}}(\boldsymbol{\epsilon}, g(\boldsymbol{\epsilon})) = \mathbf{0}$ for $\boldsymbol{\epsilon}$ in some neighbourhood of $\boldsymbol{\epsilon}'$.

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

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

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

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

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

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

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

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

B Derivation of the $\mathrm{GGN}_{\mathcal{D}}^{\texttt{MODEL}}$ 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

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

Note that we used

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

We can substitute I with

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

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:

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

we can rewrite:

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

We can thus rewrite the expression for the GGN in Equation (17) as

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

C GRADIENT COMPRESSION ABLATION

In Figure 6, we ablate different compression methods by computing the per training datapoint influence scores with compressed query (measurement) gradients, and looking at the Pearson correlation and the rank correlation to the scores compute with the uncompressed gradients. We hope to see a correlation of close to 100%, in which case the results for our method would be unaffected by compression. We find that using quantisation for compression results in almost no change to the ordering over training datapoints, even when quantising down to 8 bits. This is in contrast to the SVD compression scheme used in Grosse et al. (2023). 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. (2023). On the other hand, the diffusion training loss and other measurement functions considered in this work do not have this low-rank structure. This is because computing them requires multiple forward passes; for example, for the diffusion training loss we need to average the mean-squared error loss in Equation (2) over multiple noise samples $\epsilon^{(t)}$ and multiple diffusion timesteps. We use 8 bit quantisation with query gradient batching (Grosse et al., 2023) for all KFAC experiments throughout this work.

D DAMPING LDS ABLATIONS

We report an ablation over the LDS scores with GGN approximated with different damping factors for TRAK/D-TRAK and K-FAC influence in Figures 7 to 10.

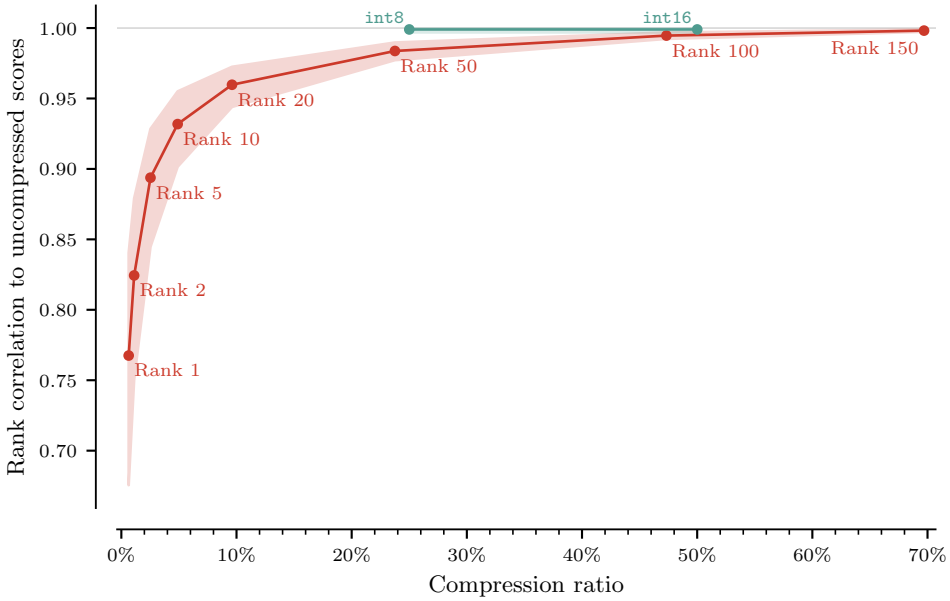


Figure 6: Comparison of gradient compression methods for the influence function approximation.

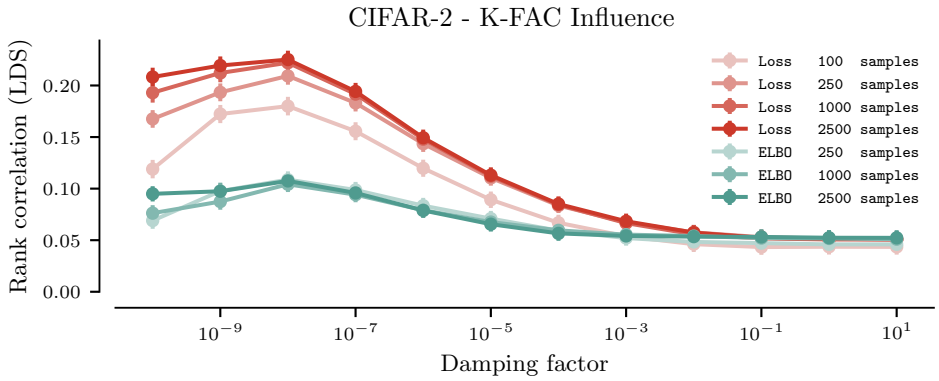


Figure 7: Effect of damping on the LDS scores for **K-FAC influence** on `CIFAR-2`.

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

F LDS Results for Probability of Sampling Trajectory

G Experimental details

G.1 Datasets

We focus on the following dataset in this paper:

`CIFAR-10` `CIFAR-10` is a dataset of small RGB images of size 32×32 Krizhevsky (2009). We use 50000 images (the train split) for training.

`CIFAR-2` For `CIFAR-2`, we follow Zheng et al. (2024) and create a subset of `CIFAR-10` of 5000 randomly subsampled images of classes `car` and `horse`, preserving an even class split.

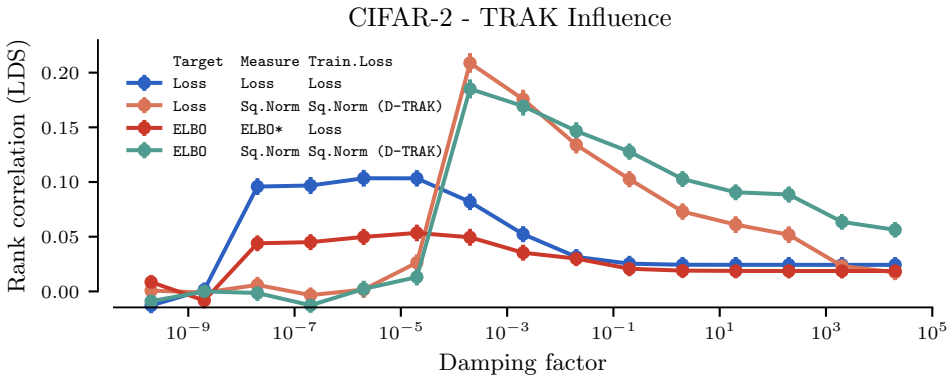


Figure 8: Effect of damping on the LDS scores for TRAK (random projection) based influence on `CIFAR-2`.

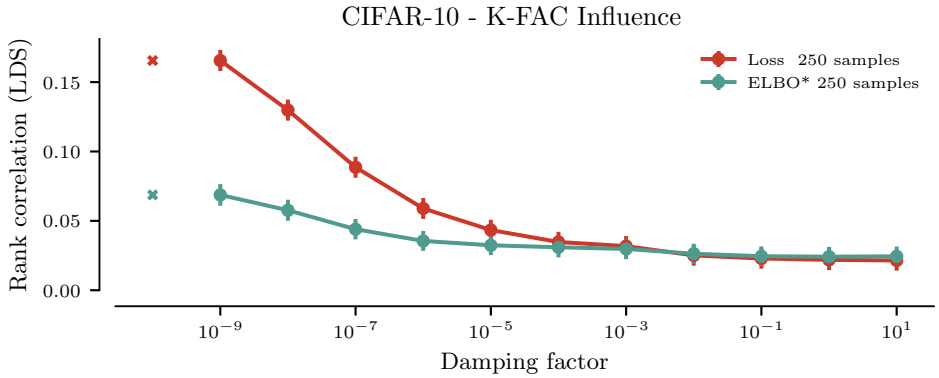


Figure 9: Effect of damping on the LDS scores for K-FAC based influence on `CIFAR-10`.

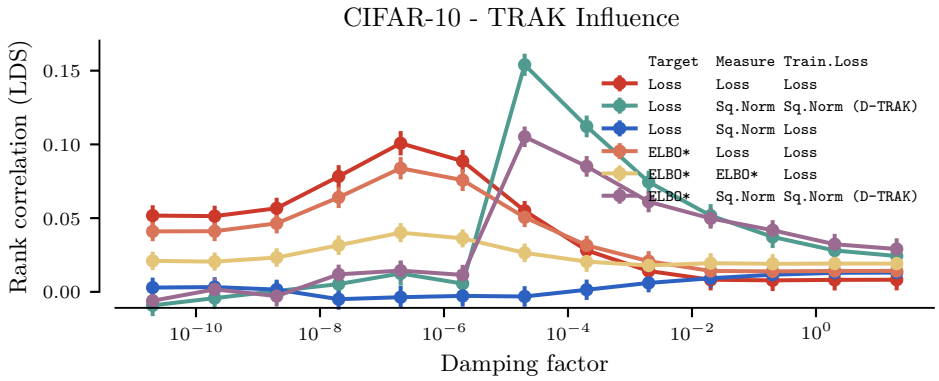


Figure 10: Effect of damping on the LDS scores for TRAK (random projection) based influence on `CIFAR-10`.

G.2 Models

For all `CIFAR` datasets, we train a regular Denoising Diffusion Probabilistic Model using a standard U-Net architecture as described for `CIFAR-10` in Ho et al. (2020). This U-Net architecture contains both convolutional and attention layers.

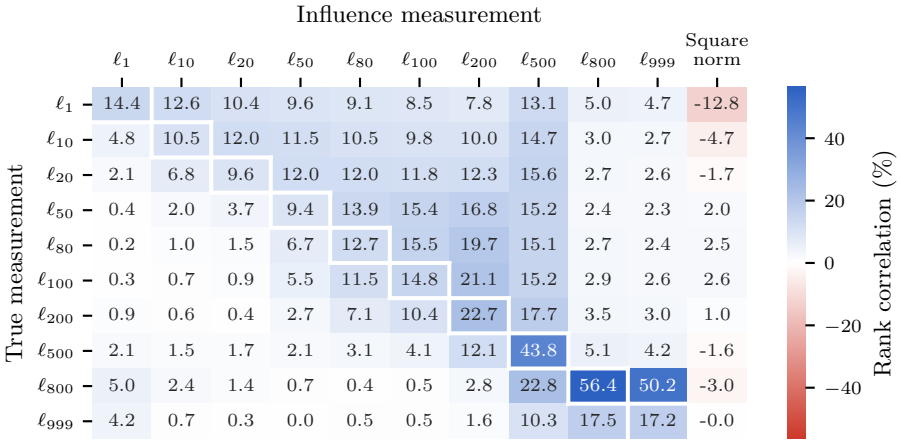


Figure 11: Rank correlation between influence estimates with different measurement functions and true measurements for losses at different diffusion timesteps on CIFAR-2.

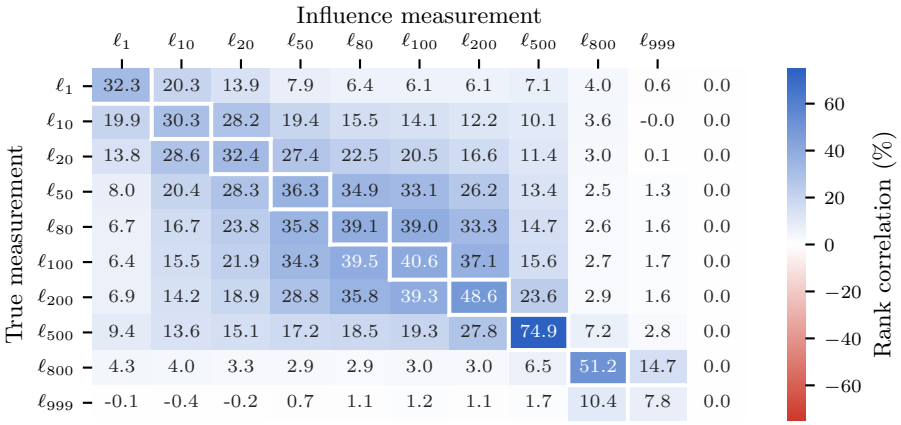


Figure 12: Rank correlation between true measurements for losses at different diffusion timesteps on CIFAR-2.

Sampling We follow the standard DDPM sampling procedure with a full 1000 timesteps to create the generated samples.

G.3 Data attribution methods

G.3.1 Damping

For all influence function-like methods (including TRAK and D-TRAK), we use damping to improve the numerical stability of the Hessian inversion. This is particularly important for methods where the Hessian approximation is at a high risk of being low-rank (for example, when using the empirical GGN in Equation (11) (which is the default setting for TRAK and D-TRAK). Concretely, before inverting the Hessian, we add a small damping factor λ to the diagonal of the Hessian approximation (for TRAK/D-TRAK, we add it directly in the projected space, as done in Zheng et al. (2024)).

TRAK For TRAK baselines, we adapt the implementation of Park et al. (2023); Georgiev et al. (2023) to the diffusion modelling setting. When running TRAK, there are several settings the authors recommend to consider: **1)** the projection dimension for the random projections, **2)** the damping factor, 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 Zheng et al. (2024).

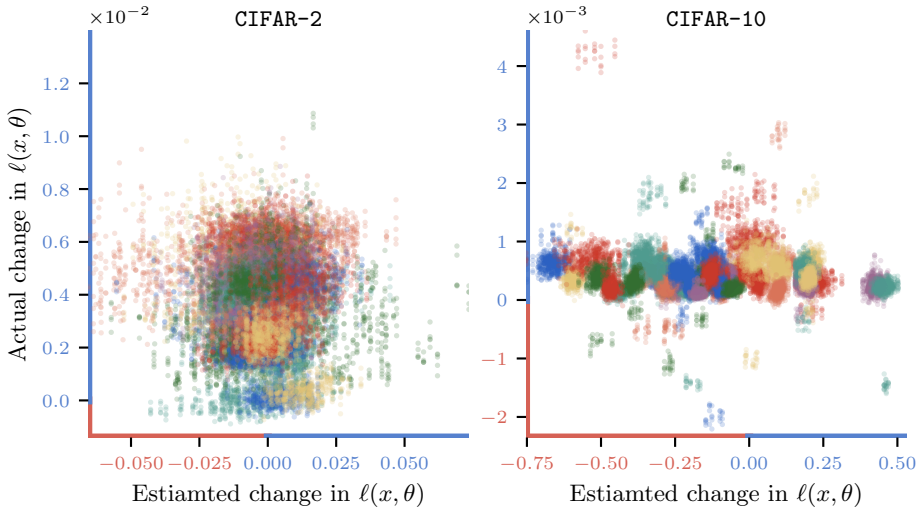


Figure 13: Change in diffusion loss ℓ in Section 2.1 when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for measurements $\ell(x, \theta)$ for 50 samples x generated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

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 18, and so we recommend using `float32` precision for these methods for diffusion models. In all experiments, we use `float32` throughout. For the damping factor, we report the sweeps over LDS scores in Figures 8 and 10, and use the best result in each benchmark, as these methods fail drastically if the damping factor is too small. The damping factor reported in the plots is normalised by the dataset size N, to match the definition of the GGN, and to make it comparable with the damping reported for other influence functions methods introduced in this paper. For non-LDS experiments, we use the best damping value from the corresponding LDS benchmark.

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

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

G.4 LDS Benchmarks

For all LDS benchmarks Park et al. (2023), we sample 100 sub-sampled datasets ($M := 100$ in Equation (13)), and we train 5 models with different random seeds ($K := 5$ in Equation (13)), each with 50% of the examples in the full dataset, for a total of 500 retrained models for each benchmark. We compute the LDS scores for 200 samples generated by the model trained on the full dataset.

G.5 Training details

For `CIFAR-10` and `CIFAR-2` we again follow the training procedure outlined in Ho et al. (2020), with the only difference being a shortened number of training iterations. For `CIFAR-10`, we train for 160000 steps (compared to 800000 in Ho et al. (2020)) for the full model, and 80000 steps for

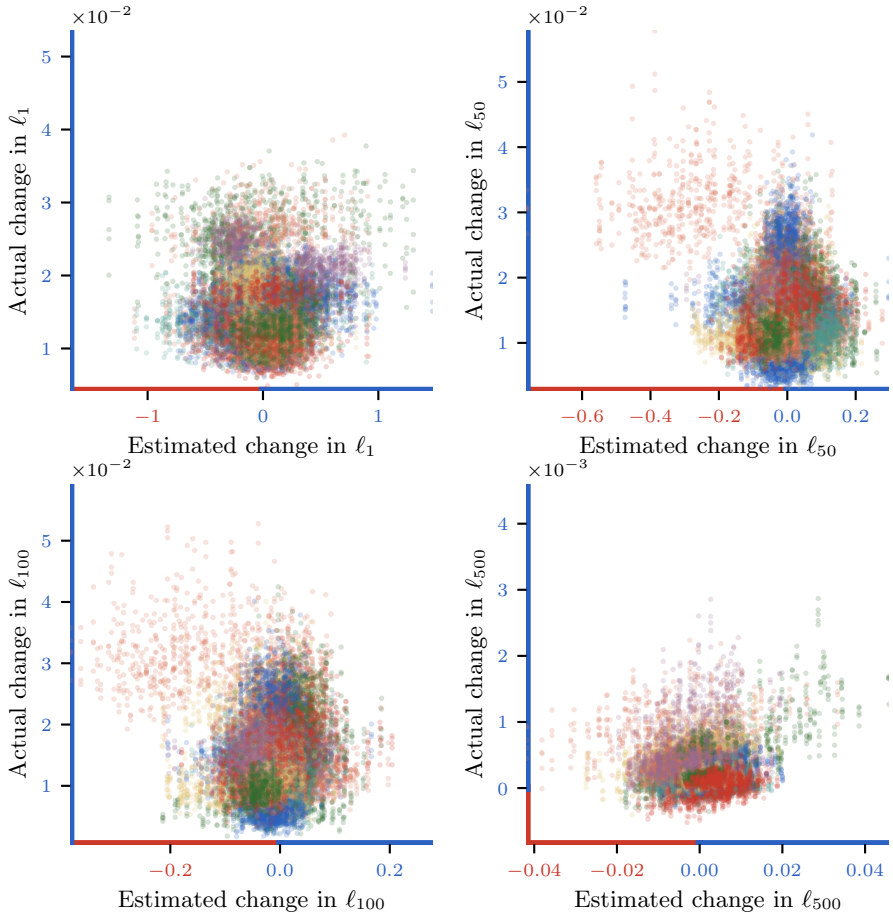


Figure 14: Change in per-diffusion-timestep losses ℓ_t when retraining with random subsets of 50% of the training data removed, as predicted by K-FAC influence (x-axis), against the actual change in the measurement (y-axis). Results are plotted for the `CIFAR-2` dataset, for measurements $\ell_t(x, \theta)$ 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 11.

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 Zheng et al. (2024), as we noticed the models trained using their procedure were somewhat significantly undertrained (some per-diffusion-timestep training losses $\ell_t(\theta, x)$ have not converged). We also use a cosine learning-rate schedule for the `CIFAR-2` models.

G.6 Retraining without top influences

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

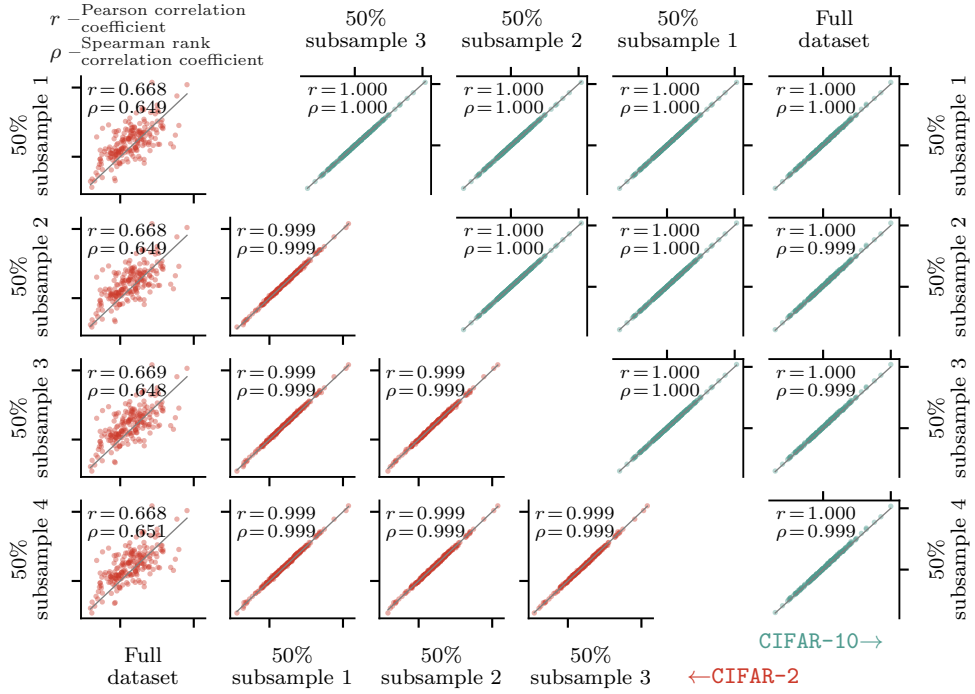


Figure 15: Correlation of the ELBO(x, θ) measurements on different data points x (samples generated from the model trained on full data), for models trained on different subsets of data (either full dataset, or randomly subsampled 50%-subset). Each subplot shows the Pearson correlation coefficient (r) and the Spearman rank correlation (ρ) for the ELBO(x, θ) measurements as measured by two models trained on different subsets of data.

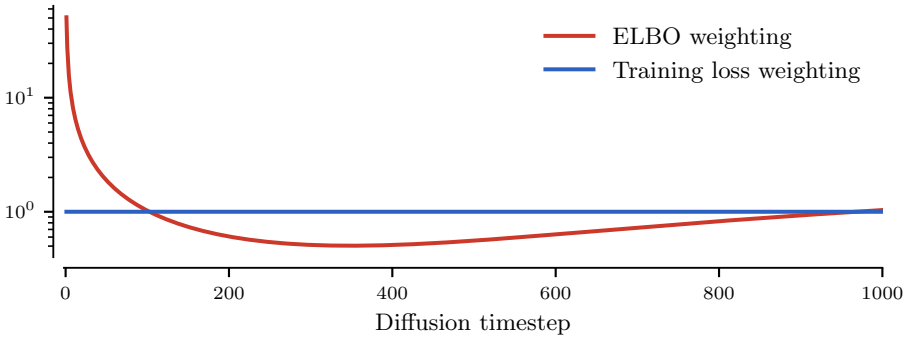


Figure 16: The diffusion loss and diffusion ELBO as formulated in (Ho et al., 2020) (ignoring the reconstruction term that accounts for the quantisation of images back to pixel space) are equal up to the weighting of the individual per-diffusion-timestep loss terms and a constant independent of the parameters. This plot illustrates the relatives difference in the weighting for per-diffusion-timestep losses applied in the ELBO vs. in the training loss.

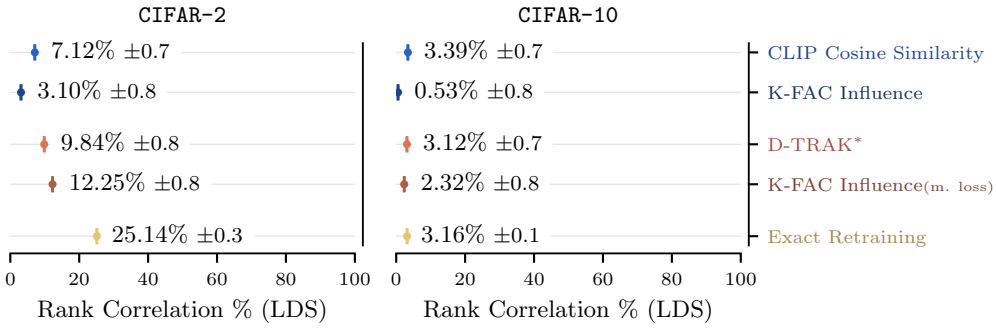


Figure 17: Linear Data-modelling Score (LDS) for the **probability of sampling trajectory**. Overall, probability of the sampling trajectory appears to be a difficult proxy for the marginal probability of sampling a given example, given that it suffers from the same issues as the ELBO on `CIFAR-2` (it's 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`). For methods marked with $*$, best results across a hyperparameter (damping factor) sweep are reported. Methods that substitute in *incorrect* measurement functions into the approximation are separated and plotted with •. "(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 '±'.

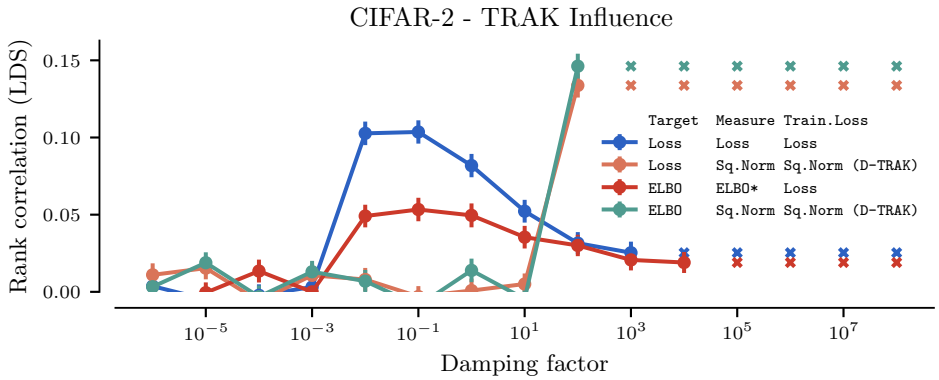


Figure 18: LDS scores on for TRAK (random projection) based influence on `CIFAR-2` when using half-precision (**`float16`**) for influence computations. Compare with Figure 8.