TOWARD EFFICIENT INFLUENCE FUNCTION: DROPOUT AS A COMPRESSION TOOL

Anonymous authors
Paper under double-blind review

ABSTRACT

Assessing the impact of training data points on machine learning models is crucial for understanding the behavior of the model and enhancing the transparency of modern models. Influence function provides a theoretical framework for quantifying the effect of individual training data points on a model's performance on given specific test data points. However, the computational cost of influence function presents significant challenges, particularly for large-scale models. In this work, we introduce a novel approach that leverages **dropout** as a gradient compression mechanism to compute the influence functions more efficiently. Our methods significantly reduces computational and memory overhead, not only during the influence function computation but also in the compression process itself. Through theoretical analysis and empirical validation, we demonstrate that using **dropout** as a compression tool in influence function computation preserves critical components of the data influence and enables its application to modern large-scale models.

1 INTRODUCTION

Large foundation models such as GPT-4 (Achiam et al., 2023), Claude (Anthropic, 2023), Gemini (DeepMind, 2023), and Llama (Touvron et al., 2023), have showcased remarkable capabilities across a variety of tasks. Despite their success, even the state-of-art models face persistent challenges, including hallucination (Huang et al., 2023; Lin et al., 2021) and the generation of toxic and biased content (Wang et al., 2023a; Abid et al., 2021). A critical factor underlying these shortcomings is the composition and quality of their training data (Park et al., 2023). Furthermore, training data not only impart knowledge to these models (Meng et al., 2022; Wang et al., 2023b), but also form the foundation of their capabilities (Mirzadeh et al., 2024). These raise a critical question: which specific data points contribute most significantly to a model's performance, and which ones negatively impact its capabilities? Addressing this question highlights the need for robust methods to evaluate the impact of individual training data points on a model's behavior and overall performance.

Influence function, a theoretical method rooted in statistics (Hampel, 1974; Law, 1986), provides a powerful tool for assessing the impact of individual training data points on a model's learned parameters and subsequently on the model's performance. Originally introduced in the context of robust statistics, it was used to assess the robustness of statistical estimators (Huber & Ronchetti, 2011). The influence function offers a framework to understand how modifications to the training dataset propagate through the model. The concept has since been adapted to deep learning (Koh & Liang, 2017; Koh et al., 2019), enabling its application to modern large-scale models. This method has been wildly used in training data selection (Xia et al., 2024; Yu et al., 2024; Hu et al., 2024), model interpretation (Grosse et al., 2023; Guo et al., 2020), data synthesizing (Li et al., 2024), and mislabel data detection (Koh & Liang, 2017; Kwon et al., 2023; Zhou et al., 2024).

Although the influence function provides a robust framework and has demonstrated promising results, their practical application is often hampered by high computational costs (Kwon et al., 2023; Zhou et al., 2024; Choe et al., 2024). Computing influence function involves calculating an inverse Hessian-vector product (iHVP) and the gradients of the loss function with respect to both training and test datasets. Since the Hessian matrix's dimensionality scales quadratically with the size of the model, and each gradient's dimensionality is the same as the model size itself, this process becomes prohibitively expensive for large-scale models. Previous methods have attempted to mit-

igate the computational burden of the influence function using iterative methods (Agarwal et al., 2017; Koh & Liang, 2017), employing an approximate version (Kwon et al., 2023), or compressing gradients (Park et al., 2023; Choe et al., 2024). However, these approaches still face challenges in scaling to modern large-scale models or adapting to diverse model architectures and structures, which further limits their practicality.

Research has shown that modern machine learning (ML) models are highly overparameterized (Fischer et al., 2024; Balaji et al., 2021), with only a small subset of parameters playing a critical role in their performance (Xue et al., 2024; Fedus et al., 2022). Furthermore, previous studies indicate that the effects of training data are closely tied to the high spectrum of the Hessian matrix, where the majority of eigenvalues are concentrated near zero, and only a few outliers deviate significantly from the bulk (Sagun et al., 2017; 2016). These findings highlight that tracking data influence does not require exhaustive computation over the entire parameter space, but can focus on a few critical directions or a small subset of parameters.

Our Contributions. We observe that the influence of particular training data points on the overall performance of a ML model can be effectively tracked through a small subset of parameters, reducing the need to consider the full parameter space. Building on this, we propose a novel dropout-based compression method to compress gradients that is straightforward to implement and scales efficiently to large-scale ML models, which significantly reduces both memory and computational complexity associated with the computation of influence function and the compression process of gradients. Through theoretical analysis and empirical experiments, we validate the effectiveness of the proposed method, demonstrating its ability to accurately capture data influence while offering computational efficiency.

2 PRELIMINARIES

We denote the input space and the output space by $\mathcal{X}$ and $\mathcal{Y}$, respectively. Let $\mathcal{D}_{\text{tr}} = \{z_{\text{tr}}^1, z_{\text{tr}}^2, \cdots, z_{\text{tr}}^n\}$ represent the training dataset, where each training data point $z_{\text{tr}}^i = (x_{\text{tr}}^i, y_{\text{tr}}^i) \in \mathcal{X} \times \mathcal{Y}$. For a given data point $z = (x, y)$ and a model with parameters $\theta \in \Theta$, let $l(y, f_\theta(x))$ denote the loss function, where $f_\theta : \mathcal{X} \to \mathcal{Y}$ is the model parameterized by θ, and $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ measures the discrepancy between the output and the ground truth. For a vector θ, the gradient of the loss function evaluated at the data point z with respect to θ is denoted as $\nabla_\theta l(y, f_\theta(x))$. Additionally, let $\mathcal{D}_{\text{val}} = (z_{\text{val}}^1, z_{\text{val}}^2, \cdots, z_{\text{val}}^m)$ denote the validation dataset, where each validation data point $z_{\text{val}}^j = (x_{\text{val}}^j, y_{\text{val}}^j) \in \mathcal{X} \times \mathcal{Y}$. Finally, we denote the number of parameters in the model by d.

2.1 INFLUENCE FUNCTION

Influence function quantifies how the model parameters change in response to upweighting a specific training data point (Law, 1986; Hampel, 1974; Koh & Liang, 2017). Formally, given an infinitesimally small $\epsilon > 0$, the upweighted empirical risk minimization problem is formulated by increasing the weight of the k-th training data point $z_{\text{tr}}^k = (x_{\text{tr}}^k, y_{\text{tr}}^k)$ in the loss function. The optimization problem is given by:

$$\theta^{(k)}(\epsilon) = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} l(y_{\text{tr}}^i, f_\theta(x_{\text{tr}}^i)) + \epsilon l(y_{\text{tr}}^k, f_\theta(x_{\text{tr}}^k)).$$

Assuming the loss function is twice-differentiable and strongly convex in θ, the influence of the k-th training data point $z_{\text{tr}}^k = (x_{\text{tr}}^k, y_{\text{tr}}^k) \in \mathcal{D}_{\text{tr}}$ on the emprircal risk minimizer θ^* is defined as the derivative of $\theta^{(k)}(\epsilon)$ at $\epsilon = 0$ (Koh & Liang, 2017):

$$\mathcal{I}_{\theta^*}(z_{\text{tr}}^k) := \frac{d\theta^{(k)}(\epsilon)}{d\epsilon}\Big|_{\epsilon=0} = -H^{-1} g_{\text{tr}}^k,$$

where $H := n^{-1} \sum_{i=1}^{n} \nabla_\theta^2 l(y_{\text{tr}}^i, f_\theta(x_{\text{tr}}^i))\big|_{\theta=\theta^*}$ is the empirical Hessian matrix and $g_{\text{tr}}^k = \nabla_\theta l(y_{\text{tr}}^k, f_\theta(x_{\text{tr}}^k))\big|_{\theta=\theta^*}$ represents the gradient of the loss function evaluated at the k-th training data point z_{tr}^k.

For the validation dataset $\mathcal{D}_{\text{val}} = (z_{\text{val}}^1, z_{\text{val}}^2, \cdots, z_{\text{val}}^m)$, the influence of the training data point z_{tr}^k on the validation loss is (Koh & Liang, 2017; Kwon et al., 2023):

$$\mathcal{I}(z_{\text{tr}}^k) := (\frac{1}{m}\sum_{j=1}^{m} g_{\text{val}}^j)^T \mathcal{I}_{\theta^*}(z_{\text{tr}}^k) = -(\frac{1}{m}\sum_{j=1}^{m} g_{\text{val}}^j)^T H^{-1} g_{\text{tr}}^k, \tag{1}$$

where $g_{\text{val}}^j = \nabla_\theta(y_{\text{val}}^j, f_\theta(x_{\text{val}}^j))\Big|_{\theta=\theta^*}$ is the gradient of the loss function evaluated at z_{val}^j.

The influence function $\mathcal{I}(z_{\text{tr}}^k)$ provides an intuitive method to evaluate how a training data point z_{tr}^k affects the performance on the validation dataset $\mathcal{D}_{\text{val}}$. In other words, the influence function quantifies whether the training data point $z_{\text{tr}}^k = (x_{\text{tr}}^k, y_{\text{tr}}^k)$ is beneficial or detrimental to the loss evaluated on $\mathcal{D}_{\text{val}}$. When the loss function is cross-entropy loss, the Hessian can be approximated with the Fisher Information Matrix (FIM), which is equivalent to the Gauss-Newton Hessian (Grosse et al., 2023; Martens, 2020; Bae et al., 2022). Note that H is not invertible if the dimension of θ exceeds the size of the training dataset n, which is common in many modern ML models. To address this issue, a damping term is added to H, i.e. using $H + \lambda I_d$ replace H, where λ is a small constant, and I_d is a $d \times d$ identity matrix.

2.2 Efficiently Calculating Influence Function

Computing the influence function faces several challenges when f_θ is a large-scale deep learning model (Kwon et al., 2023; Basu et al., 2020; Bae et al., 2022). A key obstacle is that the size of the Hessian becomes prohibitively large to compute directly, as its dimensionality scales quadratically with the number of the model parameters. This limitation is a primary reason why the influence function is not widely practical for modern large-scale ML models.

To address this challenge, several methods (Schioppa et al., 2022; Park et al., 2023) propose projecting gradients onto a low-dimensional subspace with a Gaussian random matrix (Johnson, 1984) and computing the influence function on the subspace as follows:

$$\tilde{\mathcal{I}}(z_{\text{tr}}^k) = -(\frac{1}{m}\sum_{j=1}^{m} g_{\text{val}}^j)^T P^T (PHP^T)^{-1} P g_{\text{tr}}^k, \tag{2}$$

where $P \in \mathbb{R}^{r \times d}$ is a Gaussian random matrix. Here, r represents the dimensionality of the compressed subspace. By compressing gradients into a smaller subspace using the projection matrix P, the computational and memory requirements of the computation of influence function are reduced, but it introduce additional computing and memory costs (Choe et al., 2024). The computational cost of calculating a gradient is $O(d)$ with backpropagation. In comparison, the cost of compressing the gradient into a lower-dimensional subspace using P is $O(rd)$. This makes a gradient compression process more expensive than the process of calculating a gradient. Additionally, the memory cost of the projection matrix is $O(rd)$, which can exceed the memory usage of the model itself, especially when r is large, which is to preserve expressiveness. To improve both accuracy and efficiency of influence function, further advanced method in gradient compression methods are necessary.

3 Method

To address the computational and memory challenges associated with influence function computation, we propose a novel approach that leverages dropout as a gradient compression mechanism. We demonstrate that the influence of training data on a small subset of parameters can effectively reflect its influence on the entire model. Unlike traditional gradient compression methods, which require a Gaussian random matrix as a compression matrix (Johnson, 1984), incurring significant memory and computation costs, our method randomly drops a subset of gradient entries. This technique reduces the dimensionality of the gradient and eliminates the additional memory and computation overhead associated with explicit projection matrices.

3.1 Dropout as a Compression Mechanism

Dropout is a widely used regularization technique in deep learning (Srivastava et al., 2014), where a subset of model parameters or activations is randomly set to zero during training. We apply a

similar approach to the gradient vectors during influence function computation, effectively compressing the gradient by retaining only a small subset of its components. Let g represent the gradient of the loss function with respect to the model parameters for a data point $z = (x, y)$, i.e. $g = \nabla_\theta l(y, f_\theta(x))\big|_{\theta=\theta^*}$. To compress the gradient, we randomly sample r indices corresponding to the retained components of the gradient g. Mathematically, this process is equivalent to generating a binary matrix $\tilde{I} \in \{0,1\}^{r\times d}$. Each row of $\tilde{I}$ has exactly one entry equal to 1, while all other entries are 0. Also, there is only one non-zero entry in every column. This ensures that the r-dimensional compressed gradient retains exactly r components of the original gradient. The compressed gradient $\tilde{g} \in \mathbb{R}^r$ is then computed as:

$$\tilde{g} = \tilde{I}g.$$

The binary matrix $\tilde{I}$ can be constructed by randomly sampling r rows from a d-dimensional identity matrix I_d. Each row corresponds to selecting one component of g. Specifically, $\tilde{I}_{ij} = 1$ indicated that the j-th component of the gradient is retained, while all other components are dropped.

It is important to note that the introduction of $\tilde{I}$ is primarily for theoretical analysis. In practice, we do not need to explicitly construct the matrix $\tilde{I}$ and perform matrix multiplication. This avoids unnecessary computational and memory overhead, thereby simplifying the implementation while maintaining efficiency. We randomly select a subset of entries of each gradient to do compression and compute the influence function using the compressed gradients.

Formally, replacing the original gradients with the compressed versions, the influence function is:

$$\tilde{\mathcal{I}}(z_{\text{tr}}^k) = -(\frac{1}{m}\sum_{j=1}^{m}\tilde{g}_{\text{val}}^j)^T\tilde{H}^{-1}\tilde{g}_{\text{tr}}^k = -(\frac{1}{m}\sum_{j=1}^{m}g_{\text{val}}^j)^T\tilde{I}^T(\tilde{I}H\tilde{I}^T)^{-1}\tilde{I}g_{\text{tr}}^k. \tag{3}$$

The matrix $\tilde{H}$ is the Hessian matrix/Gaussian-Newton Hessian calculated with respect to the compressed gradients, i.e. $\tilde{H} = \tilde{I}H\tilde{I}^T$.

3.2 Efficiency Comparison

Even though tradition gradient compression methods, such as random projection (Johnson, 1984) used in TRAK (Park et al., 2023), reduce the complexity of the calculation of influence function, they rely on explicit projection matrices to reduce the dimensionality of the gradient. This will introduce significant memory and computational overhead, because these methods use dense projection matrices with a memory complexity of $O(rd)$ and computational complexity dominated by matrix-vector multiplication, which is $O(rd)$. In contrast, our dropout compression method avoids the need for explicit projection matrices. By directly sampling and retaining a subset of gradient components, our method reduces memory complexity and computational cost to $O(r)$, as only the r indices and corresponding gradient entries are sampled and stored. This is significantly more efficient than the $O(rd)$ complexity of traditional methods.

Other efficient influence function computation methods, such as LiSSA (Agarwal et al., 2017) and LOGRA (Choe et al., 2024), employ stochastic iterative approaches and the Kronecker product for gradients computation, respectively. While these methods reduce the computational cost of the iHVP, they still require expensive iterative algorithms (Klochkov & Liu, 2024) or are hard to expand to all deep learning architectures (Kwon et al., 2023; Grosse et al., 2023). The comparison of computational and memory complexities with efficient influence function computation methods are detailed in Appendix C.

3.3 Error Analysis

While the compressed version in equation 3 offers a more efficient method for computing influence functions compared to equation 2, it may introduce significant errors. Intuitively, this is because the Gaussian random matrix compresses gradient information, whereas the dropout approach simply discards most gradient information. To address this concern, we theoretically analyze the error incurred by equation 3 and compare it with the error from equation 2.

Specifically, for both methods, the compression error is given by the difference $\mathcal{I}(z_{\text{tr}}^k) - \tilde{\mathcal{I}}(z_{\text{tr}}^k)$, and the spectrum norm of this error can be expressed as:

$$||\mathcal{I}(z_k) - \tilde{\mathcal{I}}(z_k)||_2 = \|(\frac{1}{m}\sum_{j=1}^{m} g_{\text{val}}^j)^T \Delta H g_{\text{tr}}^k\|_2 \leq \|\frac{1}{m}\sum_{j=1}^{m} g_{\text{val}}^j\|_2 \|\Delta H\|_2 \|g_{\text{tr}}^k\|_2, \tag{4}$$

where ΔH is defined differently for the two compression methods:

$$\Delta H = (\lambda I_d + H)^{-1} - P^T(\lambda I_r + PHP^T)^{-1}P, \tag{5}$$

for the Gaussian compression method and

$$\Delta H = (\lambda I_d + H)^{-1} - \tilde{I}^T(\lambda I_r + \tilde{I}H\tilde{I}^T)^{-1}\tilde{I}, \tag{6}$$

for the dropout compression method. Here, λ is the damping value used in both expressions, and $\|\cdot\|_2$ denotes the spectral norm of a matrix or the L_2 norm of a vector. The key factor influencing the error is the spectral norm of ΔH, i.e. $\|\Delta H\|_2$, which depends on the difference induced by compression. In the following theorems, we analyze the spectrum of ΔH and demonstrate that the error introduced by the Gaussian-based compression method is significantly larger than dropout-based compression method.

Theorem 1. *For Gaussian-based compression method in equation 2, if $\lambda I_d + P^T PH$ is invertible and the dimension of θ exceeds the size of the training dataset n, the spectral norm of the difference $\|\Delta H\|_2$ is bounded by:*

$$O(d + d^2\sigma_{max}(H)),$$

where $\sigma_{max}(H)$ denotes the largest singular value of H.

To establish this result, we simplify equation 5 using woodbury matrix identity (Harville, 1998) and leverage non-asymptotic theory of random matrices (Rudelson & Vershynin, 2010; Bai & Yin, 2008) to get the upper bound. A detailed proof is provided in Appendix E.1.

Theorem 2. *For the dropout-based compression method in equation 3, if the dimension of θ exceeds the size of training dataset n, the spectral norm of the difference $||\Delta H||_2$ is bounded by:*

$$O(\sigma_{max}(H)),$$

where $\sigma_{max}(H)$ denotes the largest singular value of H.

The proof of Theorem 2 are similar to the proof of 1, and this is detailed in Appendix E.2.

Although the error bounds are loose and derived without fully accounting for the effect of compression size on the error, they provide valuable insights into the utility of the dropout-based compression in influence function computation. The theoretical bound indicates that the dropout-based compression offers a significant advantage in terms of computational efficiency while maintaining a reasonable level of accuracy compared to other methods.

These results suggest that dropout, traditionally used as a regularization technique, can serve as a lightweight and practical tool for influence function computations, particularly in scenarios where computational resources are constrained.

4 EXPERIMENTS

In this section, we evaluate the effectiveness of our method: Use dropout as a compression tool for influence function computation, in terms of accuracy and efficiency, both of which are important in real-world data attribution tasks. Specifically, we investigate the effectiveness of our approach through two key experiments: mislabeled data detection 4.1, which uses influence function to identify mislabeled data points in a noisy training dataset, and model retraining 4.2, which identifies the most influential training data points for a model and retrains the model without those points to observe their impact on performance. To comprehensively evaluate our method, we start with relatively small experimental setups and then scale up to billion-parameter models. This allows us to assess how well our method generalizes across settings and to demonstrate its scalability. More details of the setups of experiments are provided in appendix D.1.

Table 1: Performance (AUC) of mislabeled data detection on selected GLUE benchmarks (MRPC, QNLI, SST2, and RTE) using various methods for influence function computing. The reported results are averaged over 10 independent runs. The best results are highlighted in **bold**, and the second-best results are underlined.

	rank=2				rank=8			
Method	**MRPC**	**QNLI**	**SST2**	**RTE**	**MRPC**	**QNLI**	**SST2**	**RTE**
Orig.	**0.812**	0.763	0.809	**0.616**	-	-	-	-
DataInf	0.778	0.754	0.917	0.568	0.765	0.744	0.912	0.540
LiSSA	0.651	0.504	0.509	0.505	0.663	0.502	0.480	0.499
Hessian-Free	0.681	0.630	0.822	0.527	0.679	0.559	0.764	0.502
Gaussian	0.792	**0.797**	0.926	0.586	**0.815**	0.799	0.920	0.586
Dropout (Ours)	0.800	0.796	**0.930**	0.598	**0.815**	**0.800**	**0.924**	**0.600**

4.1 Mislabeled Data Detection

Mislabeled data points often negatively impact a model's performance. It is expected that the influence values of these mislabeled data points will be larger than those of clean data points, as their inclusion tends to increase the overall loss.

In this experiment, we use four binary classification datasets from GLUE benchmark (Wang, 2018), and synthetically generate mislabeled training data points similar to (Kwon et al., 2023), flipping a binary label for 20% of randomly selected training data points to simulate the situation where a part of data points are noisy. We use the RoBERTa model (Liu, 2019) and fine-tune the model on those noisy datasets with Low-Rank Adaption (LoRA) (Hu et al., 2021) with 2-rank and 8-rank separately. As for the baselines, we investigate the performance of four efficient methods as well as the `Orig.` influence function in equation 1. We consider `LISSA` (Koh & Liang, 2017) with 10 iterations, `Hessian-free` which computes the dot product of gradients (Pruthi et al., 2020), `DataInf` that uses an approximation version of influence function (Kwon et al., 2023) and `Gaussian` which uses a Gaussian random matrix to compress gradients similar to (Park et al., 2023). Some of details of these methods are attached in Appendix B. For Gaussian-based compression and dropout-based compression, we use $r = 16$ for both 2-rank and 8-rank LoRA fine-tuning.

For evaluation metrics, we use the area under the curve (AUC) score to measure the quality of the influence function values. The AUC quantifies the ability of the influence function to distinguish between mislabeled and clean data points. Specifically, it measures the probability that a score randomly selected from a class of mislabeled data is greater than that of a class of clean data. An influence function that reliably assigns larger influence values to mislabeled data points will achieve a high AUC score, reflecting its effectiveness in identifying mislabeled examples.

Results Table 1 shows the mislabeled data detection ability comparison of the six influence computation methods when the rank of LoRA is 2 and 8. The detection ability is evaluated with AUC, and the results are averaged over 10 independent runs. The results show that `Dropout` achieves comparable detection ability to `Gaussian`, which uses a Gaussian random matrix to compress gradients, and `Orig.` which is the original version of influence function. Also, it has significantly better detection ability than `DataInf`, `LiSSA` and `Hessian-Free`. Same as (Kwon et al., 2023), we find that `Orig.` does not always have the best results. This is potentially because the gradients contain some redundant information which has some negative impacts on the performance of `Orig.`.

In terms of runtime, `Dropout` shows superior computational efficiency. For instance, on the GLUE-QNLI dataset with 8-rank LoRA, `Dropout` takes 4.36 seconds while `DataInf` take 18.79 seconds for computing the iHVP. Even though `Hessian-Free` gets rid of computing the iHVP, the performance is much worse than our method.

The time consumption of these methods across various benchmarks is provided in Appendix D.2.

Table 2: Accuracies (%) of ResNet-9 on the test dataset after removing influential training data points (remove 5%, 10%, 30% ,and 50% from the subset containing 10000 data points) from CIFAR-10 identified by various methods. The reported results are averaged over 5 independent runs. The accuracy trained on the full dataset (containing 10000 data points) is 80.50%. The best results are highlighted in **bold**, and the second-best results are underlined.

Method	5%	10%	30%	50%
Random	79.28	79.94	77.16	70.84
Hessian-Free	79.36	77.88	68.38	58.52
DataInf	78.82	78.66	69.62	56.40
LOGRA	79.60	**75.64**	69.76	64.54
Gaussian	80.18	77.36	**67.90**	56.72
Dropout	**77.88**	75.86	68.30	**55.96**

4.2 Model Retraining

The model retraining process begins by identifying the most influential data points. A specified number of highly influential data points are removed from the training dataset, and the model is then retrained on the remaining data. The performance of the retrained model is subsequently evaluated on the test dataset. A significant drop in performance after removing these data points underscores their critical role in the model's learning process and demonstrates the effectiveness of the method used to identify influential data points.

4.2.1 Small-Scale Setups

We initiate this experiment with small-scale setups: (1) ResNet-9 (He et al., 2016) with CIFAR-10, in which we train a ResNet-9 model from scratch using a randomly selected subset containing 10000 data points and evaluate on a test dataset containing 256 data points by accuracy, and (2) GPT-2 (Radford et al., 2019) with WikiText, in which we full fine-tune a GPT-2 using 2000 text samples and evaluate on a small test set containing 50 text samples using perplexity. We use influence function to compute the influential score of each training data point and rank training data points by influential scores. Then we remove the top-k or top-k percent most valuable data points from the training dataset and retrain the model multiple times without them. A larger performance decrease indicates greater effectiveness of the method in identifying the most valuable data points. On these benchmarks, we compare the performance of `Dropout` against five baselines: `Random`, which removes training data points randomly; `DataInf` (Kwon et al., 2023) which uses the approximated version of the influence function; `Hessian-Free` (Pruthi et al., 2020) which computes the dot product of gradients directly; `LOGRA` (Choe et al., 2024) which uses Kronecker product for gradients computation and compression; and `Gaussian` (Park et al., 2023) which uses a Gaussian random matrix to compress gradients. Some of the details of these methods are attached in Appendix B. For `LOGRA` we use $r_{in} = r_{out} = \sqrt{r} = 64$, and for `Gaussian` and `Dropout` we use $r = 64$ to compress the gradients. Because `LOGRA` makes use of Kronecker product to get the gradients, the compression size is much larger.

Results The retraining performance of ResNet-9 and GPT-2 are in Table 2 and Table 3, separately. The reported results are averaged over 5 independent runs. We observe that `Dropout` achieves performance comparable to traditional compression methods like `Gaussian` and the more recent one like `LOGRA` in both experiments. Moreover, it significantly outperforms `Hessian-Free` and `DataInf`.

In terms of efficiency, both `Dropout` and `Gaussian` demonstrate impressive performance in computing iHVP due to the compression of gradients. Moreover, the efficiency of gradient compression becomes crucial when dealing with large-scale models. Notably, the `Dropout` excels in efficiency during the compression process, outperforming other approaches. For example, in the GPT-2 experiment, `Dropout` requires only 9.98 seconds for gradient compression, compared to 964.65 seconds for `Gaussian`, which exceeds the time required for the gradient computation itself. Additionally,

Table 3: Test perplexity of GPT-2 after removing influential training data points (remove 150, 250, 350, and 450 from the subset containing 2000 text samples) from WikiText identified by various methods. The reported results are averaged over 5 independent runs. The perplexity of the model trained on the full dataset (containing 2000 text samples) is 14.053. The best results are highlighted in **bold**, and the second-best results are underlined.

Method	150	250	350	450
Random	15.419	18.182	23.401	23.647
Hessian-Free	15.335	18.122	23.418	23.729
DataInf	15.331	18.106	23.390	23.779
LOGRA	**15.438**	18.276	23.662	**23.962**
Gaussian	15.415	18.303	23.640	23.897
Dropout	15.420	**18.310**	**23.683**	23.959

Table 4: Test perplexity of Pythia 1.4B after removing influential training data points (remove 10%, 20%, 30%, and 40% from the training dataset containing 2000 text samples) from OpenWebText identified by various methods. The reported results are averaged over 5 independent runs. The perplexity of the model trained on the full dataset (containing 2000 text samples) is 25.23. The best results are highlighted in **bold**, and the second-best results are underlined.

Method	10%	20%	30%	40%
Random	25.52	25.91	26.28	26.49
LOGRA	**25.68**	25.89	**26.62**	26.79
Dropout	25.51	**26.05**	26.55	**26.80**

`Dropout` eliminates the extra memory overhead associated with storing the Gaussian matrix, which is a requirement for methods like `Gaussian` and `LOGRA`.

The time consumption of these methods across the experiments, including the time usage for gradients compression and iHVP computing, is detailed in Appendix D.2.

4.2.2 Large-Scale Settings

We now evaluate the practical utility of our approach for data attribution in billion-scale models. Specifically, we adopt Pythia (1.4B) (Biderman et al., 2023) and LLaMA-3.2 (1.24B) (Meta, 2024) in our experiments and conduct data attribution on a subset of OpenWebText (OWT) (Gokaslan & Cohen, 2019). As in the previous setup 4.2.1, we fine-tune the models, remove the top-k percent most influential data points from the training dataset and retrain models. A larger performance decrease indicates a more effective method for the data attribution task. It is worth to note that we use full fine-tuning in this setup too, which means the size of gradients used in the computation of influence function is the same as the size of the model itself. This makes some gradient-based data attribution methods impractical with limited computing resources, even the `Hessian-Free`, which merely computes the dot product of gradients. The large size of gradients not only leads CUDA out-of-memory (OOM) errors but also significantly increases overhead on CPUs. For methods that compress the size of gradients, `Gaussian` also become impractical due to the size of the Gaussian random matrix required for gradient compression. So, we compare the performance of `LOGRA` (Choe et al., 2024) using $r_{in} = r_{out} = \sqrt{r} = 64$, and `Dropout` with $r = 512$, both of which are practical under limited computing resources.

Results The performance of retraining Pythia and LLaMA-3.2 is presented in Table 4 and Table 5, respectively. Due to the large size of raw gradients, only `LOGRA` and `Dropout` are feasible for computing and storing gradients across all data points. We observe that `Dropout` achieves comparable performance with Pythia.

For LLaMA-3.2, we observe that the perplexity does not consistently increase as more training data points are removed. We speculate that this is because OpenWebText is included in the pre-training

Table 5: Test perplexity of LLaMA-3.2 after removing influential training data points (remove 10%, 20%, 30%, and 40% from the training dataset containing 2000 text samples) from OpenWebText identified by various methods. The reported results are averaged over 5 independent runs. The perplexity of the model trained on the full dataset (containing 2000 text samples) is 16.82. The best results are highlighted in **bold**, and the second-best results are underlined.

Method	10%	20%	30%	40%
Random	16.78	16.77	16.83	16.78
LOGRA	16.82	**16.86**	16.83	**16.93**
Dropout	**16.83**	16.83	**16.88**	16.86

dataset of LLaMA-3.2. As a result, removing these data points from the fine-tuning training dataset does not lead to a significant increase in test perplexity.

5 Analysis

In this section, We provide an additional error analysis to complement the results in section 3.3. Previously, we observed that the error upper bounds of `Gaussian` is significantly larger than that of `Dropout`. However, the analysis did not account for the compression size r, which could play a crucial role in the performance.

Intuitively, smaller compression size r will lead to greater information loss, resulting in larger errors. Notably, `Dropout` does not perform explicit information compression, instead, it simply discards information from gradients. This characteristic makes `Dropout` be more instable when the compression size r is relatively small. Therefore, it is valuable to investigate how influence function performance with `Dropout` varies with different compression size r. For this, we use mislabeled data detection as a case study.

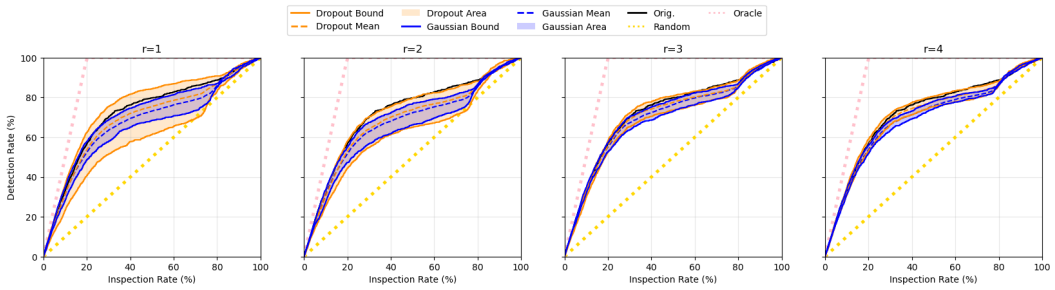

Figure 1: Mislabeled data detection on COLA (one benchmark in GLUE) with rank = 2 for LoRA fine-tuning. We compare `Orig.` (baseline) with gradient compression methods `Gaussian` and `Dropout` using different compression size r (1, 2, 3, and 4). For `Gaussian` and `Dropout`, the bounds and the average (over 5 different random seeds) detection performance are reported.

In Figure 1, we observe that the area between the bounds of the performance of `Dropout` is larger than that of `Gaussian` when the compression size r is small. This indicates that `Dropout` is less stable than `Gaussian` in small compression size setting. However, as r increasing, the performance variability of both methods narrows, and the stability becomes comparable when $r \geq 3$. Interestingly, despite the instability of `Dropout` for small r, the average performance of `Dropout` surpasses that of `Gaussian` even with a very small compression size ($r = 1$). This demonstrates the superior performance of `Dropout` in data attribution tasks, highlighting its potential as an effective gradient compression strategy.

These observations have practical implications. While smaller r values reduce memory and computation overhead, it increases performance variability. This introduces a trade-off between efficiency and robustness.

6 Related Works

Data attribution aims to quantify and understand the impact of each individual training data point on the performance of a model (Albalak et al., 2024). In (Ghorbani & Zou, 2019), the authors proposed *Data Shapley*, which quantifies the value of each training data point by leveraging Shapley values as a metric. Despite its conceptual appeal, *Data Shapley* is computationally prohibitive, particularly for modern large-scale ML models (Jia et al., 2019). Furthermore, several works such as (Ilyas et al., 2022; Park et al., 2023) proposed frameworks to do data attribution by retraining a model multiple times to evaluate the impact of some data points, thereby providing insights into their contributions to model performance. Although efforts such as (Park et al., 2023) strive to balance computational cost and effectiveness, the necessity of retraining models remains a significant drawback, especially for resource-intensive deep learning applications.

Influence function is another approach to data attribution, adapted from robust statistics (Law, 1986; Hampel, 1974), and introduced to the deep learning context in (Koh & Liang, 2017; Koh et al., 2019). It tries to answer the counterfactual question: what would happen if we remove a training point from the model. While influence function offers a theoretically grounded framework for data attribution, the high computational cost has limited its applicability to large-scale models. To mitigate the computational burden of influence function, various methods have been proposed. In (Koh & Liang, 2017; Agarwal et al., 2017), the authors introduced LiSSA, which approaches iHVP computation iteratively, reducing the cost of influence function computation. Other approaches include LOGRA (Choe et al., 2024) and EK-FAC (Grosse et al., 2023) propose using Kronecker product for gradients computation, and compress gradients using Kronecker product structure or use eigen decompositions, for efficiency. However, the Kronecker product structure cannot be universally applied to all deep learning models and eigen decompositions will be expensive in large-scale matrix. DataInf (Kwon et al., 2023) proposed an approximation of the influence function by approximating the inverse Hessian matrix. However, this method introduces errors that scale quadratically with the size of the model. This is why DataInf is less suitable for large-scale models. (Zhou et al., 2024) proposed a method which approximates the Hessian matrix using the Generated Fisher Information Matrix (GFIM). This approach relies on a strong assumption that each column of the gradient matrix is independent and has a zero mean, which often fails to hold in practice.

7 Conclusion

In this work, we demonstrate that the influence of training data on a small subset of parameters can effectively reflect its influence on the entire model. Building on this insight, we introduce **dropout** as a compression tool to enable efficient influence function computation, addressing the computational and memory challenges that hinder the application of traditional influence function in large-scale ML models. Our approach leverages the simplicity and scalability of dropout to selectively retain gradient information, thereby significantly reducing computational and memory overhead compared to methods relying on dense projection matrices such as Gaussian-based compression.

Through theoretical analysis, we demonstrated that the error upper bound of influence function with dropout-based compression is smaller than Gaussian-based compression methods. Our empirical results on mislabeled data detection and model retraining across various datasets and models validated these findings, showing that dropout achieves comparable or superior performance in data attribution while maintaining high computational efficiency. Although there is a trade-off between the compression size and performance stability, dropout-based compression method has superior average performance even in small compression size regime.

This work highlights the potential of dropout as a lightweight, efficient, and practical tool for gradient compression in influence function computation, paving the way for extending the application of influence function in large-scale artificial intelligence (AI) systems.

REFERENCES

Abubakar Abid, Maheen Farooqi, and James Zou. Persistent anti-muslim bias in large language models. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 298–306, 2021.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Naman Agarwal, Brian Bullins, and Elad Hazan. Second-order stochastic optimization for machine learning in linear time. *Journal of Machine Learning Research*, 18(116):1–40, 2017.

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, et al. A survey on data selection for language models. *arXiv preprint arXiv:2402.16827*, 2024.

Anthropic. Claude: An ai assistant by anthropic, 2023. URL `https://www.anthropic.com/`.

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

Zhi-Dong Bai and Yong-Qua Yin. Limit of the smallest eigenvalue of a large dimensional sample covariance matrix. In *Advances In Statistics*, pp. 108–127. World Scientific, 2008.

Yogesh Balaji, Mohammadmahdi Sajedi, Neha Mukund Kalibhat, Mucong Ding, Dominik Stöger, Mahdi Soltanolkotabi, and Soheil Feizi. Understanding overparameterization in generative adversarial networks. *arXiv preprint arXiv:2104.05605*, 2021.

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

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pp. 2397–2430. PMLR, 2023.

Sang Keun Choe, Hwijeen Ahn, Juhan Bae, Kewen Zhao, Minsoo Kang, Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell, Teruko Mitamura, et al. What is your data worth to gpt? llm-scale data valuation with influence functions. *arXiv preprint arXiv:2405.13954*, 2024.

Google DeepMind. Gemini: An ai model by google deepmind, 2023. URL `https://www.deepmind.com/`.

William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

Tim Fischer, Chris Biemann, et al. Large language models are overparameterized text encoders. *arXiv preprint arXiv:2410.14578*, 2024.

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

Aaron Gokaslan and Vanya Cohen. Openwebtext corpus. `http://Skylion007.github.io/OpenWebTextCorpus`, 2019.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

Han Guo, Nazneen Fatema Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. *arXiv preprint arXiv:2012.15781*, 2020.

Frank R Hampel. The influence curve and its role in robust estimation. *Journal of the american statistical association*, 69(346):383–393, 1974.

David A Harville. Matrix algebra from a statistician's perspective, 1998.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

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

Yuzheng Hu, Pingbang Hu, Han Zhao, and Jiaqi W Ma. Most influential subset selection: Challenges, promises, and beyond. *arXiv preprint arXiv:2409.18153*, 2024.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, et al. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *arXiv preprint arXiv:2311.05232*, 2023.

Peter J Huber and Elvezio M Ronchetti. *Robust statistics*. John Wiley & Sons, 2011.

Andrew Ilyas, Sung Min Park, Logan Engstrom, Guillaume Leclerc, and Aleksander Madry. Datamodels: Understanding predictions with data and data with predictions. In *International Conference on Machine Learning*, pp. 9525–9587. PMLR, 2022.

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

William B Johnson. Extensions of lipshitz mapping into hilbert space. In *Conference modern analysis and probability, 1984*, pp. 189–206, 1984.

Yegor Klochkov and Yang Liu. Revisiting inverse hessian vector products for calculating influence functions. *arXiv preprint arXiv:2409.17357*, 2024.

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

Pang Wei W Koh, Kai-Siang Ang, Hubert Teo, and Percy S Liang. On the accuracy of influence functions for measuring group effects. *Advances in neural information processing systems*, 32, 2019.

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

John Law. Robust statistics—the approach based on influence functions, 1986.

Xiaochuan Li, Zichun Yu, and Chenyan Xiong. Montessori-instruct: Generate influential training data tailored for student learning. *arXiv preprint arXiv:2410.14208*, 2024.

Stephanie Lin, Jacob Hilton, and Owain Evans. Truthfulqa: Measuring how models mimic human falsehoods. *arXiv preprint arXiv:2109.07958*, 2021.

Yinhan Liu. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364, 2019.

James Martens. New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146):1–76, 2020.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372, 2022.

Meta. Llama 3.2 model card. `https://huggingface.co/meta-llama/Llama-3.2-1B`, 2024.

Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.

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

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

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

Mark Rudelson and Roman Vershynin. Non-asymptotic theory of random matrices: extreme singular values. In *Proceedings of the International Congress of Mathematicians 2010 (ICM 2010) (In 4 Volumes) Vol. I: Plenary Lectures and Ceremonies Vols. II–IV: Invited Lectures*, pp. 1576–1602. World Scientific, 2010.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

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

Jack Sherman. Adjustment of an inverse matrix corresponding to changes in the elements of a given column or row of the original matrix. *Annu. Math. Statist.*, 20:621, 1949.

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

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Roziere, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models, 2023.

Alex Wang. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Boxin Wang, Weixin Chen, Hengzhi Pei, Chulin Xie, Mintong Kang, Chenhui Zhang, Chejian Xu, Zidi Xiong, Ritik Dutta, Rylan Schaeffer, et al. Decodingtrust: A comprehensive assessment of trustworthiness in gpt models. In *NeurIPS*, 2023a.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 2023b.

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

Fuzhao Xue, Zian Zheng, Yao Fu, Jinjie Ni, Zangwei Zheng, Wangchunshu Zhou, and Yang You. Openmoe: An early effort on open mixture-of-experts language models. *arXiv preprint arXiv:2402.01739*, 2024.

Zichun Yu, Spandan Das, and Chenyan Xiong. Mates: Model-aware data selection for efficient pretraining with data influence models. *arXiv preprint arXiv:2406.06046*, 2024.

Xinyu Zhou, Simin Fan, and Martin Jaggi. Hyperinf: Unleashing the hyperpower of the schulz's method for data influence estimation. *arXiv preprint arXiv:2410.05090*, 2024.

A LIMITATIONS

While this work demonstrates the potential of dropout as an efficient gradient compression method for influence function computation, several limitations remain to be addressed. As discussed in section 5, smaller r values in dropout, though efficient, introduce greater unstableness in performance. This unstableness can be a constraint in applications. In addition, our method does not alleviate the significant resource needs for the gradient computation. Computing gradients for all data points, particularly in large-scale models and datasets, remains a bottlenecks. This limitation highlights the need for optimizations to make influence function methods more resource-efficient.

B EFFICIENT METHODS FOR INFLUENCE FUNCTION

LiSSA Agarwal et al. (2017) proposed an iterative method for computing iHVP $(H + \lambda I_d)^{-1}v$, which was later utilized by Koh & Liang (2017) to calculate the influence function. For $s_0 = v$, LiSSA recursively computes the following equation: $s_{i+1} = v + (I_d - (H + \lambda I_d))s_i$. Agarwal et al. (2017) proved that if $H + \lambda I_d \preceq I_d$, s_i will converge to iHVP $(H + \lambda I_d)^{-1}v$, as i increases. Then, the iHVP could be approximated as:

$$s_i \approx (H + \lambda I_d)^{-1}v, \tag{7}$$

and the influence function could be calculated by this approximation:

$$\mathcal{I}(z_{\text{tr}}^k) = -s_i^T g_{\text{tr}}^k. \tag{8}$$

Here $g_{\text{tr}}^k = \nabla_\theta l(y_{\text{tr}}^k, f_\theta(x_{\text{tr}}^k))\big|_{\theta=\theta^*}$ is the gradient of the loss function calculated at the k-th training data point with respect to the parameters of model, and $v = \frac{1}{m}\sum_{j=1}^m g_{\text{val}}^j = \frac{1}{m}\sum_{j=1}^m \nabla_\theta l(y_{\text{val}}^j, f_\theta(x_{\text{val}}^j))\big|_{\theta=\theta^*}$ is the average of gradients of the loss function calculated at evaluation data points with respect to the parameters of model.

DataInf Kwon et al. (2023) proposed an approximate version of the influence function. The key approximation in DataInf involves swapping the order of matrix inversion and the averaging in $(H + \lambda I_d)^{-1}$. Using this approximation, the inverse Hessian matrix becomes:

$$(\frac{1}{n}\sum_k g_{\text{tr}}^k g_{\text{tr}}^{kT} + \lambda I_d)^{-1} \approx \frac{1}{n}\sum_k (g_{\text{tr}}^k g_{\text{tr}}^{kT} + \lambda I_d)^{-1} \tag{9}$$

$$= \frac{1}{n\lambda}\sum_k (I_d - \frac{g_{\text{tr}}^k g_{\text{tr}}^{kT}}{\lambda + g_{\text{tr}}^{kT} g_{\text{tr}}^k}), \tag{10}$$

where g_{tr}^k is the gradient of the loss function calculated at the k-th training data point with respect to the parameters of the model. The Sherman-Morrison formula (Sherman, 1949) is utilized to compute the matrix inversion in equation 10. Based on this approximation, the influence function can be computed efficiently, reducing the operation to linear complexity.

LOGRA Grosse et al. (2023); Choe et al. (2024) proposed using Kronecker product to approximate gradients and Choe et al. (2024) compresses gradients make use of Kronecker product structure. For the l-th layer of a deep learning model with parameter θ_l, let h_l represent the output and g_l represent the pre-activated output of the l-th layer. The gradient of loss function evaluated on $z = (x, y)$ with respect to θ_l is given as the following:

$$\nabla_{\theta_l} l = h_{l-1} \otimes \nabla_{g_l} l, \tag{11}$$

where $\otimes$ represents the Kronecker product. LOGRA (Choe et al., 2024) imposes an additional Kronecker product structure on the projection matrix P as follows:

$$P\nabla_{\theta_l} l = (P_{in} \otimes P_{out})(h_{l-1} \otimes \nabla_{g_l} l) \tag{12}$$

$$= P_{in} h_{l-1} \otimes P_{out} \nabla_{g_l} l, \tag{13}$$

where $P_{in} \in \mathbb{R}^{r_{in} \times d_{in}}$, $P_{out} \in \mathbb{R}^{r_{out} \times d_{out}}$. In equation 13, LOGRA first projects forward and backward activations onto low-dimensional spaces with P_{in} and P_{out} respectively, and then reconstructs projected gradient directly from these projected activations. It is important to note that $d_{in} = d_{out} = \sqrt{d}$ and $r_{in} = r_{out} = \sqrt{r}$, making it straightforward to use relatively large compression size r.

Table 6: Comparison of complexity of influence function computation between `Orig`, `LiSSA`, `DataInf` and gradient compression methods (includes `Gaussian`, `Dropout` and `LOGRA`). In this case, the number of parameters in the model is d, the number of data points is n, and the compression size is r.

Method	Computational Complexity	Memory Complexity
Orig.	$O(nd^2 + d^3)$	$O(d^2)$
LiSSA	$O(nd^2)$	$O(d^2)$
DataInf	$O(nd)$	$O(d)$
Gradient Compression Methds	$O(nr^2 + r^3)$	$O(r^2)$

Table 7: Comparison of complexity of performing compression between `Gaussian`, `LOGRA`, and `Dropout`. In this case, the number of parameters in the model is d, the number of data points is n, and the compression size is r.

Method	Computational Complexity	Memory Complexity
Gaussian	$O(nrd)$	$O(rd)$
LOGRA	$O(n\sqrt{rd})$	$O(\sqrt{rd})$
Dropout	$O(nr)$	$O(r)$

C Complexity Comparison

Table 6 presents a comparison of the computational and memory complexities of influence function computation among `Orig`, `LiSSA`, `DataInf` and gradients compression methods with compression size r, such as `Dropout`, `Gaussian` and `LOGRA`. While gradient compression methods cannot reduce the complexity to linear, the compression size r can be very small, making these methods efficient in practice.

Although the influence function computation complexities for gradient compression methods are the same, the complexity of compression operations themselves differ. Table 7 provides a comparison of `Dropout` with `Gaussian` and `LOGRA` in terms of the complexity of compression process.

Because of the structure of Kronecker product used in LOGRA, it also reduces the gradient computations process from $O(nd)$ to $O(nr)$, where n is the number of data points, d is the number of parameters in the model and r is the compression size.

D Experiments

D.1 Details of Experiments

For each methods, we set the damping term in influence function as $\lambda_l = 0.1 \times (nd_l)^{-1} \sum_{i=1}^{n} \nabla_{\theta_l} l_i^T \nabla_{\theta_l} l_i$ for layer l, where θ_l represents the parameters of the l-th layer, $\nabla_{\theta_l} l_i$ represents the gradient of the loss function calculated at the i-th data point with respect to θ_l, and d_l represents the number of parameters in this layer. We use one H100GPU with 80GB VRAM for all our evaluation experiments.

For model training, we use hyperparameters in Table 8 for each experiments.

D.2 More Results

In this section, we include more results of our experiments. Table 9 contains the average time usage for computing iHVP in the experiments of mislabeled data detection. Table 10 and Table 11 present the average time usage for gradients compression and iHVP computation in experiments involving the retraining of ResNet-9 and GPT-2. respectively. It is evident that the dropout-based method not

Table 8: Hyperparameters used for model training in our experiments.

	RoBERTa	**ResNet-9**	**GPT-2**	**Pythia**	**LLaMA-3.2**
Optimizer	AdamW	SGD-M	AdamW	AdamW	AdamW
LR Scheduler	Linear	Linear	None	None	None
Learning Rate	$3e-4$	0.4	$5e-5$	$2e-5$	$2e-5$
Weight Decay	None	$5e-4$	0.01	0.01	0.01
Batch Size	32	64	64	16	16
Sequence Length	None	N/A	256	128	128
Epochs	10	24	3	1	1

Table 9: Average time usage (seconds) of computing iHVP in mislabeled data detection tasks.

	rank=2				**rank=8**			
Method	**MRPC**	**QNLI**	**SST2**	**RTE**	**MRPC**	**QNLI**	**SST2**	**RTE**
Orig.	564.62	661.88	718.92	452.04	-	-	-	-
DataInf	8.35	10.26	10.27	5.57	14.59	18.79	18.93	9.79
LiSSA	43.69	53.59	53.82	29.45	79.29	101.36	101.74	56.80
Gaussian	3.50	4.31	4.31	2.39	3.56	4.45	4.40	2.48
Dropout (Ours)	3.47	4.28	4.28	2.37	3.52	4.36	4.35	2.42

only achieves superior efficiency in iHPV computation process, but is also highly efficient in the compression process itself.

E Proof of Theorems

We first introduce some theorems which will be used in the proof.

The Woodbury matrix identity (Theorem 3) allows cheap computation of inverses.

Theorem 3. *Given a square invertible $n \times n$ matrix A, an $n \times k$ matrix U, and a $k \times n$ matrix V, let B be an $n \times n$ matrix such that $B = A + UV$. Then, assuming $(I_k + VA^{-1}U)^{-1}$ is invertible, we have:*

$$B^{-1} = A^{-1} - A^{-1}U(I_k + VA^{-1}U)^{-1}VA^{-1}.$$

Rudelson & Vershynin (2010); Bai & Yin (2008) introduced a theorem about convergence of extreme singular values and prove that the largest singular value σ_{max} of a $k \times n$ random matrix converges to $O(\sqrt{k})$.

Theorem 4. *Let $A = A_{k,n}$ be a $k \times n$ random matrix whose entries are independent copies of some random variable with zero mean, unit variance, and finite fourth moment. Suppose that the dimensions k and n grow to infinity while the aspect ratio $\frac{n}{k}$ converges to some number $\kappa \in (0, 1]$. Then, almost surely*

$$\frac{1}{\sqrt{k}}\sigma_{min}(A) \longrightarrow 1 - \sqrt{\kappa}, \quad \frac{1}{\sqrt{k}}\sigma_{max}(A) \longrightarrow 1 + \sqrt{\kappa}.$$

We will make use of Theorem 3 and 4 to prove Theorem 1 and 2.

E.1 Proof of Theorem 1

We assume $\lambda I_d + P^T PH$ is invertible, taking advantage of woodbury matrix identity (Theorem 3), ΔH could be expressed as:

$$\begin{aligned}\Delta H &= (\lambda I_d + H)^{-1} - P^T(\lambda I_r + PHP^T)^{-1}P \\ &= (\lambda I_d + H)^{-1} - \frac{1}{\lambda}P^T P - \frac{1}{\lambda}P^T PH(\lambda I_d + P^T PH)^{-1}P^T P.\end{aligned}$$

Table 10: Average time usage (seconds) of compression gradients and computing iHVP in the experiment involving retraining ResNet-9.

	DataInf	**Gaussian**	**Dropout**
Gradients Compression	-	94.02	5.95
iHVP	41.32	4.92	4.75

Table 11: Average time usage (seconds) of compression gradients in the experiment involving retraining GPT-2.

	Gaussian	**Dropout**
Gradients Compression	964.65	9.98

Accordingly, the spectrum norm of ΔH is:

$$\|\Delta H\|_2 = \|(\lambda I_d + H)^{-1} - \frac{1}{\lambda}P^T P - \frac{1}{\lambda}P^T PH(\lambda I_d + P^T PH)^{-1}P^T P\|_2,$$

and make use of basic properties of norm and singular value, we could get:

$$\begin{aligned}\|\Delta H\|_2 &\leq \|(\lambda I_d + H)^{-1}\|_2 + \frac{1}{\lambda}\|P^T P\|_2 + \frac{1}{\lambda}\|P^T PH\|_2\|(\lambda I_d + P^T PH)^{-1}\|_2\|P^T P\|_2 \\ &= \frac{1}{\sigma_{min}(\lambda I_d + H)} + \frac{\sigma_{max}(P^T P)}{\lambda} + \frac{\sigma_{max}(P^T PH)\sigma_{max}(P^T P)}{\lambda\sigma_{min}(\lambda I_d + P^T PH)} \\ &\leq \frac{1}{\sigma_{min}(\lambda I_d + H)} + \frac{\sigma_{max}(P^T P)}{\lambda} + \frac{\sigma_{max}(P^T P)\sigma_{max}(H)\sigma_{max}(P^T P)}{\lambda\sigma_{min}(\lambda I_d + P^T PH)}. \end{aligned} \tag{14}$$

Because the dimension of θ exceeds the size of training dataset n, the Hessian/Gauss-Newton Hessian matrix H cannot be full rank, which means $\sigma_{min}(\lambda I_d + H)$ and $\sigma_{min}(\lambda I_d + P^T PH)$ are λ. In addition, because of the theorem about convergence of extreme singular values (Theorem 4), we have:

$$\sigma_{max}(P^T P) \leq \sigma_{max}(P^T)\sigma_{max}(P) \leq d.$$

Thus, from Equation 14, we have:

$$\|\Delta H\|_2 \leq \frac{1+d}{\lambda} + \frac{d^2\sigma_{max}(H)}{\lambda^2} \propto d + d^2\sigma_{max}(H).$$

As a result, $\|\Delta H\|_2$ is bounded by $O(d + d^2\sigma_{max}(H))$. □

E.2 Proof of Theorem 2

We have a binary matrix $\tilde{I}$ provided in Equation 3, where $\lambda I_d + \tilde{I}^T\tilde{I}H$ is invertible. Using woodbury matrix identity (Theorem 3), ΔH can be expressed as:

$$\begin{aligned}\Delta H &= (\lambda I_d + H)^{-1} - \tilde{I}^T(\lambda I_r + \tilde{I}H\tilde{I}^T)^{-1}\tilde{I} \\ &= (\lambda I_d + H)^{-1} - \frac{1}{\lambda}\tilde{I}^T\tilde{I} - \frac{1}{\lambda}\tilde{I}^T\tilde{I}H(\lambda I_d + \tilde{I}^T\tilde{I}H)^{-1}\tilde{I}^T\tilde{I}.\end{aligned}$$

Similar to the proof of Theorem 1, the spectrum of ΔH can be expressed as:

$$\|\Delta H\|_2 = \|(\lambda I_d + H)^{-1} - \frac{1}{\lambda}\tilde{I}^T\tilde{I} - \frac{1}{\lambda}\tilde{I}^T\tilde{I}H(\lambda I_d + \tilde{I}^T\tilde{I}H)^{-1}\tilde{I}^T\tilde{I}\|_2,$$

and be bounded by:

$$\begin{aligned}\|\Delta H\|_2 &\leq \|(\lambda I_d + H)^{-1}\|_2 + \frac{1}{\lambda}\|\tilde{I}^T\tilde{I}\|_2 + \frac{1}{\lambda}\|\tilde{I}^T\tilde{I}H\|_2\|(\lambda I_d + \tilde{I}^T\tilde{I}H)^{-1}\|_2\|\tilde{I}^T\tilde{I}\|_2 \\ &= \frac{1}{\sigma_{min}(\lambda I_d + H)} + \frac{\sigma_{max}(\tilde{I}^T\tilde{I})}{\lambda} + \frac{\sigma_{max}(\tilde{I}^T\tilde{I}H)\sigma_{max}(\tilde{I}^T\tilde{I})}{\lambda\sigma_{min}(\lambda I_d + \tilde{I}^T\tilde{I}H)} \\ &\leq \frac{1}{\sigma_{min}(\lambda I_d + H)} + \frac{\sigma_{max}(\tilde{I}^T\tilde{I})}{\lambda} + \frac{\sigma_{max}(\tilde{I}^T\tilde{I})\sigma_{max}(H)\sigma_{max}(\tilde{I}^T\tilde{I})}{\lambda\sigma_{min}(\lambda I_d + \tilde{I}^T\tilde{I}H)}. \end{aligned} \tag{15}$$

Given the particular form of the binary matrix $\tilde{I}$, it is easy to verify that $\tilde{I}^T\tilde{I}$ is a binary diagonal matrix. Accordingly, we have $\sigma_{max}(\tilde{I}^T\tilde{I}) = 1$. In addition, as previously described, $\sigma_{min}(\lambda I_d + H)$ and $\sigma_{min}(\lambda I_d + \tilde{I}^T\tilde{I}H)$ are λ. Thus, equation 15 yields:

$$\|\Delta H\|_2 \leq \frac{2}{\lambda} + \frac{\sigma_{max}(H)}{\lambda^2} \propto \sigma_{max}(H).$$

As a result, $\|\Delta H\|_2$ is bounded by $O(\sigma_{max}(H))$. □