LoRIF: Pushing the Scalability Frontier of Training Data Attribution

Shuangqi Li
EPFL, Switzerland
shuangqi.li@epfl.ch

Hieu Le
UNC Charlotte, USA
hle40@charlotte.edu

Jingyi Xu
Stony Brook University, USA
jingyixu@cs.stonybrook.edu

Mathieu Salzmann
EPFL, Switzerland
mathieu.salzmann@epfl.ch

Abstract

Training data attribution (TDA) estimates how training examples affect a model's predictions. Influence function methods are a theoretically grounded family of TDA methods and exploit gradients. To overcome the scalability challenge arising from gradient computation, the most popular strategy is random projection (e.g., TRAK, LoGRA). However, this still faces two bottlenecks when scaling to large training sets and high-quality attribution: *(i)* storing and loading projected per-example gradients for all N training examples, where query latency is dominated by I/O; and *(ii)* forming the $D \times D$ inverse Hessian approximation, which costs $O(D^2)$ memory. Both bottlenecks scale with the projection dimension D, yet increasing D is necessary for attribution quality—creating a quality–scalability tradeoff. We introduce **LoRIF** (**Lo**w-**R**ank **I**nfluence **F**unctions), which exploits low-rank structures of gradient to address both bottlenecks. First, we store rank-c factors of projected per-example gradients rather than full matrices, reducing storage and query-time I/O from $O(D)$ to $O(c\sqrt{D})$ per layer per sample. Second, we use truncated SVD with the Woodbury identity to approximate the inverse Hessian term in an r-dimensional subspace, reducing memory from $O(D^2)$ to $O(Dr)$. Across models from 124M to 70B parameters trained on corpora of up to 3.8M examples, LoRIF achieves up to 20$\times$ storage reduction and query-time speedup compared to LoGRA, while matching or exceeding its attribution quality. LoRIF makes gradient-based TDA practical at frontier scale. Code is available at https://github.com/doub7e/LoRIF.

1 Introduction

Training data attribution (TDA) asks how training examples affect a model's behavior on a query input. As language models are increasingly trained and adapted on billions to trillions of tokens, this question is becoming central to model debugging, data curation, safety auditing, and diagnosing data contamination or poisoning. Yet despite years of methodological progress, gradient-based TDA has remained impractical at frontier scale. Existing methods have demonstrated gradient-based attribution on large language models [Grosse et al., 2023, Choe et al., 2024, Chang et al., 2024], but repeated-query use remains prohibitively costly in practice. Recomputing training gradients incurs substantial latency, while caching them trades this computation for degraded attribution quality, persistent storage and I/O.

The gap between theory and practice is not for lack of a good attribution formula. *Influence functions* [Koh and Liang, 2017] provide a theoretically-grounded approach to TDA: they approximate

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

leave-one-out retraining via the use of gradients and inverse Hessian. However, exact influence computation is too expensive for modern neural networks: explicitly forming curvature matrices is memory-prohibitive, and iterative solvers require repeated Hessian–vector products over the training set. Scalable variants such as TRAK [Park et al., 2023], LoGRA [Choe et al., 2024], and TrackStar [Chang et al., 2024] therefore store projected per-example gradients and use a damped Gauss–Newton Hessian approximation. These methods define the scalable frontier of IF-style TDA: after a one-time indexing pass, they can answer many attribution queries. However, their quality is constrained by the feasible projection dimension.

Specifically, the bottlenecks of projection-based TDA methods are twofold. First, **gradient storage and I/O**: storing projected per-example gradients for all N training examples requires $O(ND)$ space, where D is the effective projection dimension (the total dimension of the stored gradient per example); at query time, loading these gradients dominates latency (around 96% of the query time). Second, **inverse Hessian scaling**: forming and storing the $D \times D$ inverse Hessian approximation costs $O(D^2)$ memory. Both bottlenecks scale with D, yet increasing D is key to high attribution quality. This creates a quality-scalability tradeoff: accurate attribution (large D) is impractical, while tractable attribution (small D) degrades quality.

We introduce LoRIF, which substantially mitigates this tradeoff by exploiting a key property: **neural-network gradients have low effective rank** (see Section 2.3), both within layers (per-example) and in the aggregate matrix across examples. To address the storage and I/O bottleneck, LoRIF stores each projected per-example gradient as a rank-c factorization, reducing storage and query-time I/O from $O(D)$ to $O(c\sqrt{D})$ per layer. To address the $D \times D$ memory bottleneck, we compute a truncated SVD of the gradient matrix and exploit the Woodbury identity to compute the curvature term in an r-dimensional subspace without forming a dense $D \times D$ inverse—reducing memory cost from $O(D^2)$ to $O(Dr)$. Together, these approximations enable increasing D (improving quality) while keeping both storage/I/O and curvature costs tractable.

We evaluate LoRIF on GPT2-small, OLMo-3-7B, and Apertus-70B, spanning 124M–70B parameters and 233K–3.8M attribution examples. Compared to LoGRA, LoRIF achieves 2.3–20$\times$ lower gradient storage and 1.3–20$\times$ lower query latency in comparable-quality settings, and up to 2.2$\times$ higher tail-patch scores on Apertus-70B with lower storage and latency. Overall, our contributions are:

- We identify the projection-dimension bottleneck in scalable IF-style TDA: larger projected dimensions improve attribution quality but make gradient storage, query-time I/O, and curvature approximation expensive.
- We introduce rank-c storage for projected per-example gradients, reducing per-example, per-layer storage from $O(D)$ to $O(c\sqrt{D})$.
- We introduce a rank-r spectral representation of projected curvature, reducing per-layer memory from $O(D_\ell^2)$ to $O(D_\ell r)$ through the Woodbury identity.
- To our knowledge, we provide the first demonstration of reusable influence-function-based attribution at the scale of a 70B model over 3.8M SFT examples, without recomputing training-example gradients for each query batch.

2 Background and Related Work

2.1 Influence Functions

A theoretically-grounded way to estimate the impact of a training sample on a model's prediction consists of defining an influence function based on the model's gradients. Below, we provide some background on this approach, before delving into the attempts that have been made to improve the scalability of this strategy.

Classical influence functions. Given a model with parameters θ^* trained to minimize an empirical risk, the influence of a training sample x_{tr} on a test sample x_{te} can be approximated as [Koh and Liang, 2017]

$$\mathcal{I}(x_{tr}, x_{te}) = g_{te}^\top H^{-1} g_{tr}, \tag{1}$$

where g_{tr} and g_{te} are the gradients of the loss with respect to the model parameters on the training and test samples, and H is the Hessian of the training loss. This formulation approximates how the

model's prediction on the test sample x_{te} would change if the training sample x_{tr} were infinitesimally upweighted.

Gauss-Newton Hessian approximation. For deep models, the Hessian matrix H is very large and typically positive *semi*-definite, making it intractable to compute and invert. As such, recent works have approximated the Hessian using the damped Gauss-Newton matrix [Teso et al., 2021, Bae et al., 2022, Grosse et al., 2023, Choe et al., 2024, Chang et al., 2024]

$$H \approx G^\top G + \lambda I, \tag{2}$$

where $G \in \mathbb{R}^{N \times D}$ is the matrix of gradients (or projected gradients, as discussed below) for all N training samples, and $\lambda > 0$ is a damping term. With this approximation, the influence function in Equation (1) becomes:

$$\mathcal{I}(x_{tr}, x_{te}) \approx g_{te}^\top (G^\top G + \lambda I)^{-1} g_{tr}. \tag{3}$$

However, calculating Eq. (2) in the original parameter space is often intractable, as the gradient can have billions of dimensions.

Inverse Hessian-vector product. Instead of explicitly forming the Hessian or its approximation, one can directly approximate the inverse Hessian-vector products (iHVP). Early iHVP approximations include stochastic Neumann-series estimators such as LiSSA [Agarwal et al., 2017] and Krylov-subspace methods such as Arnoldi iteration [Schioppa et al., 2022]. Grosse et al. [2023] proposes using (E)K-FAC for iHVP approximation, leveraging the Kronecker structure of gradient matrices. ASTRA [Wang et al., 2025a] further reduces the iterations needed for accurate iHVP estimation by using EKFAC as a preconditioner for iterative Neumann-series updates. DataInf [Kwon et al., 2024] derives a closed-form approximation for influence computation, with experiments focused on LoRA-tuned models. Beyond curvature computation, storing or recomputing per-example training gradients remains a separate bottleneck for repeated attribution queries over large datasets.

Random projection for tractability. Even with the Gauss-Newton Hessian and iHVP approximations, working in the original parameter space is a computational and storage bottleneck at scale, requiring storing or recomputing high-dimensional per-example gradients. Wojnowicz et al. [2016] and Park et al. [2023] project the (flattened) gradients to a lower-dimensional space using random projection matrices, making Eq. (2) tractable. LoGRA [Choe et al., 2024] and TrackStar [Chang et al., 2024] reduce the memory cost of random projection matrices by applying *two-sided* random projections: for each linear layer with input dimension I and output dimension O, the gradients are projected from $\mathbb{R}^{O \times I}$ to $\mathbb{R}^{d_2 \times d_1}$ using separate random matrices $P_{in} \in \mathbb{R}^{I \times d_1}$ and $P_{out} \in \mathbb{R}^{O \times d_2}$. For an *effective projection dimension* $D = d_1 \cdot d_2$, the memory cost of the random projection matrices for each layer is reduced from $O(DIO)$ to $O(d_1 I + d_2 O)$. Specifically, LoGRA realizes the two-sided projection through a LoRA-like auxiliary module and stores the full D-dimensional projected gradient for each example.

2.2 Other TDA Families

TDA methods span several families that differ in the attribution quantity they estimate and in the artifacts they require. Simulation-based and subset-valuation methods, such as Data Shapley [Ghorbani and Zou, 2019], datamodeling [Ilyas et al., 2022], and Simfluence [Guu et al., 2023], estimate counterfactual data effects using many trained models, subset indicators, or learned simulators of training behavior. Trajectory-aware methods, such as TracIn [Pruthi et al., 2020], SGD-Influence [Hara et al., 2019], HyDRA [Chen et al., 2021], and SOURCE [Bae et al., 2024], use checkpoints, update histories, or approximate unrolling to account for the training trajectory. Representation-based approaches include representer point selection [Yeh et al., 2018], which decomposes predictions using training-point representations, and RepSim [Hanawa et al., 2020], which retrieves examples by representation similarity.

These families are important parts of the TDA landscape, but they answer related yet different attribution questions, require different training artifacts, or face significant scalability barriers. Appendix A gives a broader taxonomy for positioning LoRIF.

2.3 Low-Rank Gradients in Neural Networks

LoRIF exploits two complementary low-rank phenomena observed in neural-network gradients:

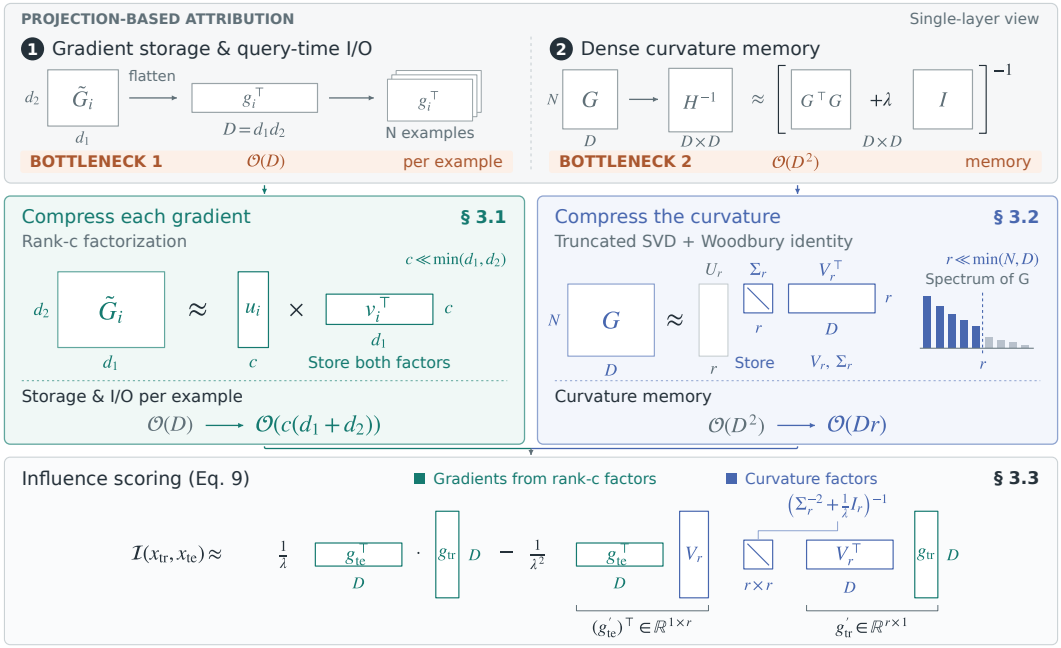


Figure 1: **Method Overview.** (**Top**): LoGRA-style attribution faces two scaling bottlenecks: storing projected gradients with $\mathcal{O}(D_\ell)$ space per example per layer, and forming the dense $D_\ell \times D_\ell$ curvature approximation with $\mathcal{O}(D_\ell^2)$ memory. LoRIF targets each bottleneck with a complementary low-rank step. (**Mid-left**): Rank-c factorization compresses each projected per-example gradient, reducing persistent storage and query-time I/O. (**Mid-right**): A rank-r truncated SVD of the per-layer training-gradient matrix G reduces curvature memory to $\mathcal{O}(D_\ell r)$. (**Bottom**): Combining both compressions and applying the Woodbury identity result in a more scalable influence scoring method, making larger projection dimensions practical under storage, I/O and memory constraints.

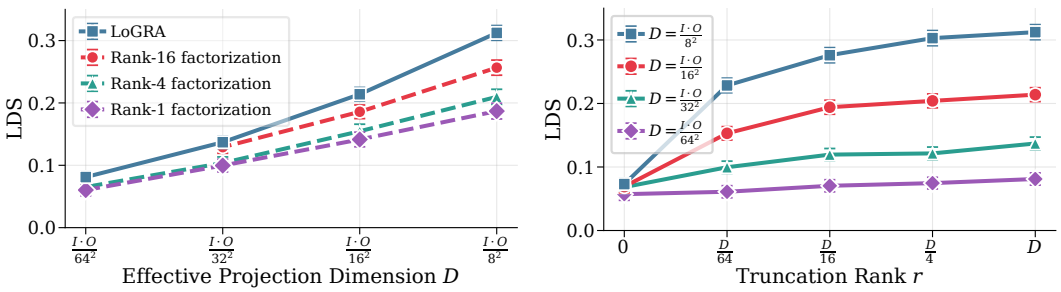
Per-example gradients have low effective rank. In practice, it has been shown that the weight gradients of a linear layer are often well approximated by low-rank matrices, a phenomenon leveraged by gradient compression methods [Wang et al., 2018, Vogels et al., 2019, 2020] and memory-efficient training [Zhao et al., 2024, Gooneratne et al., 2020, Liang et al., 2024]. A related concept is Low-Rank Adaptation (LoRA) [Hu et al., 2022, Hao et al., 2024, Liu et al., 2024], which enables parameter-efficient fine-tuning by constraining the weight updates to low-rank subspaces.

Cross-example gradients have low effective rank. Empirical studies of deep-network Hessians frequently report a *spiked* spectrum: a small number of dominant directions together with a large bulk of near-zero eigenvalues [Sagun et al., 2016, Wu et al., 2020]. Related work reports similar spectral concentration for Fisher/Fisher-variant curvature [Karakida et al., 2019, Singh et al., 2021]. Together, these findings suggest that the gradient Gram matrix $G^\top G$, as well as G itself, can have low *effective* rank in practice.

LoRIF applies these observations to two distinct representations: the stored, token-aggregated projected gradient of each training example and the cross-example matrix used for curvature estimation. This differs from structuring the projection operator, as in LoGRA, or retaining the exact activation/backpropagation factors along a training trajectory [Wang et al., 2025c].

3 Method: LoRIF

LoRIF follows the standard projection-based attribution (e.g., TRAK, LoGRA) pipeline but addresses the two scalability bottlenecks: (1) gradient storage and query-time I/O, and (2) inverse Hessian scaling. Our method follows three steps: first, we compute and store per-example gradients using low-rank factorization (§3.1); second, we compute an inverse Hessian approximation via truncated



(a) Attribution quality vs. D.

(b) Attribution quality vs. r.

Figure 2: **Effects of low-rank approximations on attribution quality.** Linear Datamodeling Score (LDS) is the Spearman correlation between additive attribution predictions and outputs of models retrained on data subsets; higher is better. (**Left**): LoRIF trades a larger effective projection dimension D against rank-c factorization; LoGRA (no factorization) is compared against rank-c factorization, varying D by setting $d_1 = \frac{I}{f}$ and $d_2 = \frac{O}{f}$ for $f \in \{64, 32, 16, 8\}$ (so $D = \frac{I \cdot O}{f^2}$). (**Right**): truncated-SVD curvature approximation approaches the full-rank baseline with $r \ll D$; $r = 0$ means discarding the curvature information and reduces to dot product; rank factorization is **not** used. Both experiments use GPT2-small on WikiText-103.

SVD and the Woodbury identity (§3.2); third, we compute influence scores at query time using the stored low-rank factors and the truncated inverse Hessian(§3.3). Figure 1 illustrates our method.

3.1 Per-Example Gradients

Consider a model with L linear layers. Layer ℓ has input dimension I_ℓ, output dimension O_ℓ, and weight matrix $W_\ell \in \mathbb{R}^{O_\ell \times I_\ell}$. Following LoGRA, we apply two-sided random projections with matrices $P^\ell_{in} \in \mathbb{R}^{I_\ell \times d^\ell_1}$ and $P^\ell_{out} \in \mathbb{R}^{O_\ell \times d^\ell_2}$. For training sample i at layer ℓ, let $X^\ell_i \in \mathbb{R}^{T \times I_\ell}$ be the input activations and $\delta Y^\ell_i \in \mathbb{R}^{T \times O_\ell}$ the output gradients across T tokens. The projected gradient matrix is then

$$\tilde{G}^\ell_i = (P^\ell_{out})^\top \nabla_{W_\ell} \ell_i P^\ell_{in} = (\delta Y^\ell_i P^\ell_{out})^\top (X^\ell_i P^\ell_{in}) \in \mathbb{R}^{d^\ell_2 \times d^\ell_1}. \tag{4}$$

We define the *effective projection dimension* for layer ℓ as $D_\ell = d^\ell_1 \cdot d^\ell_2$, and the flattened per-example gradient vector $g^\ell_i := \mathrm{vec}(\tilde{G}^\ell_i) \in \mathbb{R}^{D_\ell}$. We then denote by $G^\ell \in \mathbb{R}^{N \times D_\ell}$ the matrix whose i-th row is $(g^\ell_i)^\top$. In what follows, we omit the layer index ℓ when the context is clear.

Rather than storing $\tilde{G}^\ell_i$ directly, we store a rank-c factorization

$$\tilde{G}^\ell_i \approx u^\ell_i (v^\ell_i)^\top, \qquad u^\ell_i \in \mathbb{R}^{d^\ell_2 \times c}, \quad v^\ell_i \in \mathbb{R}^{d^\ell_1 \times c}. \tag{5}$$

computed via a few block power iterations. This reduces storage per layer per sample from $d^\ell_1 d^\ell_2$ floats to $c(d^\ell_1 + d^\ell_2)$ floats.

Figure 2a validates this approximation and guides our choice of c. Even the most aggressive compression ($c = 1$) retains meaningful attribution quality. Moreover, for a fixed storage budget, increasing D improves quality more than increasing c: quadrupling D (which doubles storage for the low-rank factorization) yields larger gains than quadrupling c at fixed D (which quadruples storage). This indicates that lower c should be used for better cost efficiency, prioritizing larger D over a higher factorization rank.

Grosse et al. [2023] also leverage the low-rank structure of gradients, but for a different purpose: they store rank-32 approximations of preconditioned *query* gradients in order to fit more of them in memory and amortize the cost of recomputing training gradients. We instead factorize *training* gradients with lower ranks, prioritizing storage reduction of the N-example database over per-gradient fidelity.

3.2 Inverse Hessian Approximation

To efficiently compute Eq. (3), projection-based methods explicitly form and store $K = (G^\top G + \lambda I)^{-1} \in \mathbb{R}^{D \times D}$ in the projected space, avoiding slower iterative iHVP solvers and gradient recom-

putation. However, forming this matrix and inverting it costs $O(ND^2 + D^3)$ time and $O(D^2)$ space, which becomes infeasible as D grows. Following Choe et al. [2024], Chang et al. [2024], we form K individually for each layer (a block-diagonal Hessian approximation), but this alone still limits D_ℓ to roughly 10^5 in practice.

This constraint conflicts with the quality gains from larger D. As Figure 2a shows, attribution quality improves substantially as D increases, suggesting continued gains beyond the feasible range. For example, setting $d_1 = I/4$ and $d_2 = O/4$ gives $D = IO/16$, which can exceed 1.5×10^5 for a GPT2-small layer with $I = 768$ and $O = 3072$. Since $H \in \mathbb{R}^{D \times D}$, storing H alone requires more than 80GB in FP32, and even more for computing the inverse. To unlock these gains, we need an inverse Hessian approximation that scales better than quadratically in D.

To further reduce the cost of forming and storing $(G^\top G + \lambda I)^{-1}$, we leverage a rank-$r$ truncated SVD of $G \in \mathbb{R}^{N \times D}$ (layer index ℓ omitted for brevity), i.e.,

$$\begin{aligned} G &\approx U_r \Sigma_r V_r^\top, \\ U_r \in \mathbb{R}^{N \times r},\ &\Sigma_r \in \mathbb{R}^{r \times r},\ V_r \in \mathbb{R}^{D \times r}. \end{aligned} \tag{6}$$

with $r \ll \min(N, D)$. We perform randomized SVD [Halko et al., 2011] once after collecting all training gradients, reconstructing rows of G batch-by-batch from the stored low-rank factors without materializing G in memory.

The low-rank Hessian approximation is then $H \approx V_r \Sigma_r^2 V_r^\top + \lambda I$. Applying Woodbury identity gives

$$(V_r \Sigma_r^2 V_r^\top + \lambda I)^{-1} = \frac{1}{\lambda} I - \frac{1}{\lambda^2} V_r \underbrace{\left(\Sigma_r^{-2} + \frac{1}{\lambda} I_r \right)^{-1}}_{r \times r \text{ diagonal}} V_r^\top. \tag{7}$$

This reduces the time complexity of approximating H^{-1} from $O(ND^2 + D^3)$ to $O(NDc + NDr)$ and the space complexity from $O(D^2)$ to $O(Dr)$. We store only $V_r \in \mathbb{R}^{D \times r}$ and the r singular values in Σ_r—reducing the memory cost of storing the H^{-1} approximation from $O(D^2)$ to $O(Dr)$.

We evaluate the quality-efficiency tradeoff of this approximation by varying r and D. In particular, Figure 2b reports attribution quality (LDS) as a function of r for different effective dimensions D, compared against the full-rank baseline from Eq. (3). These results confirms that attribution quality approaches the full-rank level for $r \ll D$. Additional evidence and analysis of the spectrum concentration of G are provided in Appendix E.2.

3.3 Influence Scores

For a given query example x_{te}, we denote the corresponding flattened projected gradient as g_{te}. Similarly, we use x_{tr} and g_{tr} to denote any example in the training set and its corresponding flattened projected gradient.

At query time, we load the stored Σ_r, V_r (Eq. (6)) and form g_{te} from the stored low-rank factors from Eq. (5) in GPU memory. For the training gradients g_{tr}, we load their low-rank factors from high-performance storage (e.g., NVMe) to GPU memory batch-by-batch to form them on the fly during the influence computation. We project the gradients into the r-dimensional subspace as

$$g'_{tr} = V_r^\top g_{tr} \in \mathbb{R}^r, \quad g'_{te} = V_r^\top g_{te} \in \mathbb{R}^r. \tag{8}$$

Substituting Eq. (6) and Eq. (7) into Eq. (3), the influence becomes

$$\mathcal{I}(x_{tr}, x_{te}) \approx \frac{1}{\lambda} g_{te}^\top g_{tr} - \frac{1}{\lambda^2} (g'_{te})^\top \left(\Sigma_r^{-2} + \frac{1}{\lambda} I_r \right)^{-1} g'_{tr}. \tag{9}$$

The first dot product can be approximated directly from the rank-c factors as $\langle u_{te}^\top u_{tr},\, v_{te}^\top v_{tr} \rangle_F$, costing $O(c^2(d_1 + d_2))$ instead of $O(d_1 d_2)$. The second term costs $O(r)$ since the middle matrix is diagonal.

Let us consider scoring N_q query examples against all N training examples. LoRIF achieves speedup from:

- **Reduced I/O.** LoRIF loads $c(d_1 + d_2)$ floats per training sample per layer instead of $d_1 d_2$—a compression ratio of $d_1 d_2 / c(d_1 + d_2) \approx \min(d_1, d_2)/2$.
- **Reduced computation.** LoGRA requires $O(D^2 N_q)$ for applying H^{-1} to query gradients, plus $O(DNN_q)$ for pairwise dot products. LoRIF requires $O(rD(N + N_q))$ to project all gradients via V_r, plus $O(c^2(d_1 + d_2)NN_q + rNN_q)$ for influence computation via Eq. (9). When $N_q \ll N$ and r is relatively small (e.g., $r \approx N_q$), the projection cost $O(rD(N + N_q))$ is comparable to LoGRA's scoring cost $O(DNN_q)$. Further, when c is small (e.g., $c = 1$), the per-pair cost drops substantially from $O(D)$ to $O(c^2(d_1 + d_2) + r)$. Most of our experiments in Section 4 fall into this case.

Figure 3 confirms this analysis with $N = 233$k, $N_q = 1$k, $D/r = 16$, and D being up to 37k for some layers. LoGRA is I/O-bounded (96% of 211s spent loading gradients). Rank-1 factorization alone reduces I/O by $\sim 40\times$, cutting the total time to 11s. Adding the truncated SVD approximation further reduces computation, yielding 7s in total—a $30\times$ speedup over LoGRA.

We acknowledge that both the rank-1 factorization and the truncated SVD introduce approximation errors. However, as we will show in Section 4, LoRIF enables recovering or exceeding LoGRA's attribution quality while enjoying the efficiency gains because we can increase D beyond the feasible range of LoGRA.

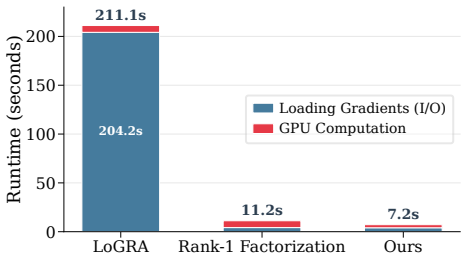


Figure 3: **Time spent in loading gradients and GPU computation.** Query-time latency breakdown for different methods, with the same effective projection dimension $D = \frac{I \cdot O}{8^2}$ and truncation rank $r = \frac{D}{16}$ for each layer of GPT2-small. "Ours" combines rank-1 factorization and truncated SVD.

LoRIF is implemented on top of the LoGRA pipeline, but the two techniques we introduce are not specific to LoGRA. The rank-c factorization can be applied after any method that produces projected per-example gradient matrices for linear layers, including other efficient gradient projection or sketching methods [Schioppa, 2024, Hu et al., 2025]. Similarly, the truncated-SVD curvature approximation can be used by methods that form a Gauss–Newton-style curvature term from stored projected gradients. For example, TrackStar also requires per-example gradient computation and uses a related Hessian approximation, making both LoRIF components compatible with its pipeline.

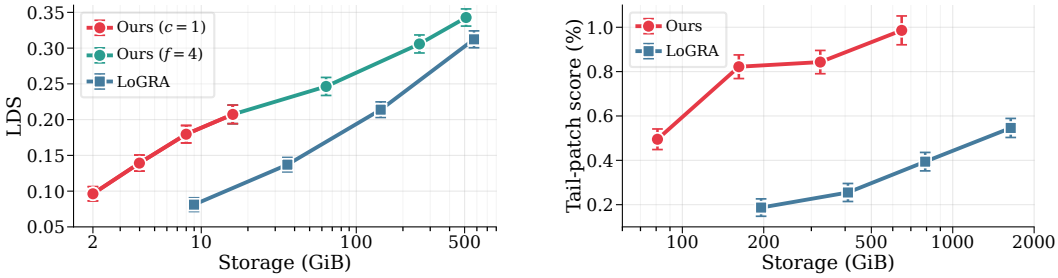
4 Experiments

We evaluate LoRIF on three axes: (1) storage reduction, (2) query-time speedup, and (3) attribution quality compared to baseline methods. We describe our choices for baselines, metrics, models, and datasets in the following subsection.

4.1 Experimental Setup

Baselines. Our primary comparisons are against influence-function-style methods that share LoRIF's final-checkpoint attribution setting: LoGRA [Choe et al., 2024], TrackStar [Chang et al., 2024], GradDot [Charpiat et al., 2019, Pruthi et al., 2020]. LoGRA and TrackStar use projected per-example gradients with curvature-aware scoring, while GradDot uses projected gradient dot products without Hessian preconditioning. We also include EK-FAC [Grosse et al., 2023] and RepSim [Hanawa et al., 2020] as contextual baselines: EK-FAC represents parameter-space influence with a different recomputation profile, and RepSim represents representation-based retrieval.

Metrics. We evaluate attribution quality using LDS [Park et al., 2023], a standard retraining-based proxy that measures how well attribution scores predict output changes under training-data subsampling, and tail-patch score [Chang et al., 2024], which measures whether the top-k retrieved proponents increase the query target probability after one additional training step. LDS is used for GPT2-small, where repeated subset training is feasible; tail-patch is used for larger scales and its alignment with LDS is studied in Figure 5. For efficiency, we report training-gradients storage and query latency, where latency includes loading the stored gradient representations and scoring the full query batch: 487 queries for GPT2-small and 1,000 for each large-model setting. Preprocessing times and query-time amortization are reported in Appendix C.



(a) LDS vs. storage on GPT2-small.

(b) Tail-patch score vs. storage on OLMo3-7B.

Figure 4: **LoRIF (Ours) vs. LoGRA under different storage budgets.** (**Left**): Attribution quality measured by LDS on GPT2-small. We vary the storage cost by choosing different effective projection dimensions D, where $D = \frac{I \cdot O}{f^2}$, and factorization ranks c (only for LoRIF). For LoGRA, we choose $f \in \{64, 32, 16, 8\}$. For LoRIF, we use $f \in \{32, 16, 8, 4\}$ when $c = 1$, and $c \in \{1, 4, 16, 32\}$ when $f = 4$. (**Right**): Attribution quality measured by tail-patch score on OLMo3-7B. For LoGRA, we choose $f \in \{360, 256, 180, 128\}$. For LoRIF, we use $f \in \{128, 64, 32, 16\}$ and $c = 1$.

Models and datasets. We evaluate on three settings spanning 124M–70B parameters and 232K–3.8M attribution examples: GPT2-small [Radford et al., 2019] on WikiText-103, OLMo3-7B [Olmo et al., 2025] on Dolci Think SFT, and Apertus-70B [Apertus et al., 2025] on its public SFT data. For GPT2-small, validation sequences are used as queries. For OLMo-3-7B and Apertus-70B, we sample 1,000 prompts from Nemotron Cascade SFT [Wang et al., 2025b], generate model responses as queries, and compute gradients only on assistant tokens.

Additional details on the baselines, metrics, models, and datasets are given in Appendix B.

4.2 Main Results

We first evaluate whether LoRIF improves the quality–efficiency frontier of projection-based influence-function methods. All $\pm$ values in the main comparison tables are 95% query-bootstrap confidence-interval half-widths. Figure 4 gives the overall quality–storage tradeoff, while Table 1 and 2 report detailed results with corresponding configurations.

On GPT2-small, LoRIF improves the Pareto frontier relative to LoGRA and TrackStar. At the high-quality end, LoRIF achieves the highest LDS among the projected-gradient methods while using less storage. In tighter storage regimes, LoRIF matches or nearly matches LoGRA's LDS using roughly 8–9$\times$ less storage and up to $5.6\times$ lower latency. EK-FAC achieves higher LDS, but it requires repeated gradient recomputation and is therefore orders-of-magnitude slower; we include it as a contextual parameter-space IF baseline rather than a scalable query-time indexing method.

The same pattern holds at larger scale. On OLMo-3-7B, LoRIF can either nearly match LoGRA's tail-patch score with $20.3\times$ lower storage and $22.0\times$ lower latency, or use a larger effective projection dimension to achieve a $1.8\times$ higher tail-patch score while still using less storage and latency. On Apertus-70B, LoRIF improves tail-patch by 1.5–$2.2\times$ over LoGRA, again with lower storage and query latency. These results support the central mechanism of LoRIF: low-rank storage makes larger projected dimensions feasible, and the larger effective attribution space yields better quality under realistic storage and I/O constraints.

Ablations and Approximation Diagnostics LoRIF combines two low-rank approximations: rank-c projected-gradient storage and truncated-SVD curvature approximation. Appendix D shows that they address complementary bottlenecks: removing truncated SVD makes high-dimensional settings run out of memory, while removing rank factorization restores the projected-gradient storage and I/O cost. Appendix E further analyzes approximation behavior across model scales and discusses how we choose (f, c, r).

Direct retraining validation. On a controlled 1,024-example GPT2-small task, we evaluate 50 randomly chosen individual removals over 50 paired seeds. LoRIF attains a mean per-query Spearman correlation of 0.3966 with the measured LOO effects, compared with 0.3648 for LoGRA. Appendix G

Table 1: **Main comparison on GPT2-small.** We control the persistent storage cost and the query latency for a query batch by varying the hyperparameters (f, c, r). Note that EK-FAC stores curvature statistics only and recomputes training gradients. Bold values are the best observed values within each storage panel.

Method	f	c	r	LDS ↑	Storage ↓	Latency ↓
Contextual baselines						
EK-FAC	–	–	–	0.3936 ± 0.013	2.1 GiB	*20 hr*
RepSim	–	–	–	0.0103 ± 0.009	340.7 MiB	0.15 s
High storage regime						
GradDot	8	–	–	0.0576 ± 0.010	574.9 GiB	**209.8 s**
TrackStar	8	–	–	0.2623 ± 0.012	574.9 GiB	241.1 s
LoGRA	8	–	–	0.3123 ± 0.012	574.9 GiB	211.1 s
LoRIF	4	32	2^{14}	$\mathbf{0.3428 \pm 0.012}$	**511.0 GiB**	287.1 s
Medium storage regime						
TrackStar	16	–	–	0.1797 ± 0.011	143.7 GiB	64.9 s
LoGRA	16	–	–	$\mathbf{0.2139 \pm 0.011}$	143.7 GiB	56.3 s
LoRIF	4	1	2^{12}	0.2073 ± 0.013	**16.6 GiB**	**25.5 s**
Low storage regime						
TrackStar	32	–	–	0.1340 ± 0.010	35.9 GiB	18.2 s
LoGRA	32	–	–	0.1370 ± 0.010	35.9 GiB	16.4 s
LoRIF	16	1	2^{11}	$\mathbf{0.1392 \pm 0.011}$	**4.1 GiB**	**2.93 s**

Table 2: **Large-scale attribution results.** Tail-patch score is reported on OLMo-3-7B and Apertus-70B, where repeated subset retraining for LDS is infeasible.

Method	f	c	r	Tail-patch (%) ↑	Storage ↓	Latency ↓
OLMo-3-7B						
RepSim	–	–	–	0.144 ± 0.04	**16.99 GiB**	**7.26 s**
GradDot	128	–	–	0.411 ± 0.04	1638.6 GiB	781.4 s
LoGRA	128	–	–	$\underline{0.546 \pm 0.04}$	1638.6 GiB	782.1 s
LoRIF	128	1	2^8	0.495 ± 0.05	<u>80.9 GiB</u>	<u>35.6 s</u>
LoRIF	16	1	2^8	$\mathbf{0.986 \pm 0.07}$	647.4 GiB	469.7 s
Apertus-70B						
LoGRA	512	–	–	0.177 ± 0.04	1815.1 GiB	1087.9 s
LoRIF	256	1	2^9	0.262 ± 0.03	**333.7 GiB**	**222.5 s**
LoRIF	64	1	2^9	$\mathbf{0.391 \pm 0.04}$	1334.7 GiB	843.8 s

gives the protocol, confidence intervals, and a full-corpus top/bottom removal experiment assessing data-selection utility.

4.3 Top-1 Retrieval Evaluation

The preceding metrics evaluate either retraining-based predictive agreement or short-horizon causal effect of an additional training update. We also evaluate whether the retrieved training examples are useful for inspection. To avoid relying only on selected qualitative examples, we conduct a systematic LLM-as-a-judge evaluation of the top-1 retrieved example from LoRIF and LoGRA on 1,000 random queries per large model. The judge (Claude Haiku 4.5) assigns each retrieved example a relevance score from 1 to 5, where 1 indicates a completely irrelevant retrieval and 5 indicates a nearly identical or directly corresponding retrieval. Table 3 summarizes

Table 3: **LLM-as-a-judge top-1 retrieval evaluation.** The judge scores on a 1–5 relevance scale. Preference reports the fraction of queries where one method is judged better or two are tied.

	Avg. relevance ↑		Preference (%)		
Model	**LoRIF**	**LoGRA**	**LoRIF**	**LoGRA**	**Tie**
OLMo-3-7B	3.04	2.49	35.1	8.0	56.9
Apertus-70B	2.91	2.24	40.8	9.1	50.1

the results; the full score distributions, judge prompts, negative cases, RepSim comparison, and safety-auditing case study are given in Appendix H.

5 Limitations

LoRIF reduces persistent gradient storage and repeated query-time latency, but constructing the attribution index still requires a gradient pass over the attribution corpus. For modern LLMs trained for only a few epochs, this preprocessing cost can still be substantial. We mitigate this by truncating sequence lengths, but developing more efficient methods is a promising direction for future research. Also, our large-scale experiments evaluate attribution for supervised fine-tuning data rather than full pre-training corpora. Finally, while our experiments analyses provide scalable evidence of attribution quality, empirical validation at true pre-training scale remains future work.

6 Conclusion

We introduced LoRIF, a scalable influence-function-style method for training data attribution. LoRIF does not rely on a new low-rank primitive; rather, it shows where standard low-rank structure should be exploited in projected-gradient influence methods. Rank-factorized projected-gradient storage addresses the persistent storage and query-time I/O bottleneck, while truncated-SVD curvature approximation addresses the memory bottleneck of curvature-aware scoring. Together, these components make larger effective projection dimensions practical, improving the storage–latency–quality frontier across models from GPT2-small to Apertus-70B. These results suggest that gradient-based TDA can be made practical for large-scale LLM auditing and debugging when the attribution index is reused across many queries.

References

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

P. Apertus, A. Hernández-Cano, A. Hägele, A. H. Huang, A. Romanou, A.-J. Solergibert, B. Pasztor, B. Messmer, D. Garbaya, E. F. Ďurech, et al. Apertus: Democratizing open and compliant llms for global language environments. *arXiv preprint arXiv:2509.14233*, 2025.

J. Bae, N. Ng, A. Lo, M. Ghassemi, and R. B. Grosse. If influence functions are the answer, then what is the question? *Advances in Neural Information Processing Systems*, 35:17953–17967, 2022.

J. Bae, W. Lin, J. Lorraine, and R. B. Grosse. Training data attribution via approximate unrolling. *Advances in Neural Information Processing Systems*, 37:66647–66686, 2024.

T. A. Chang, D. Rajagopal, T. Bolukbasi, L. Dixon, and I. Tenney. Scalable influence and fact tracing for large language model pretraining. *arXiv preprint arXiv:2410.17413*, 2024.

G. Charpiat, N. Girard, L. Felardos, and Y. Tarabalka. Input similarity from the neural network perspective. *Advances in Neural Information Processing Systems*, 32, 2019.

Y. Chen, B. Li, H. Yu, P. Wu, and C. Miao. Hydra: Hypergradient data relevance analysis for interpreting deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 7081–7089, 2021.

S. K. Choe, H. Ahn, J. Bae, K. Zhao, M. Kang, Y. Chung, A. Pratapa, W. Neiswanger, E. Strubell, T. Mitamura, et al. What is your data worth to gpt? llm-scale data valuation with influence functions. *arXiv preprint arXiv:2405.13954*, 2024.

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

M. Gooneratne, K. C. Sim, P. Zadrazil, A. Kabel, F. Beaufays, and G. Motta. Low-rank gradient approximation for memory-efficient on-device training of deep neural network. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3017–3021. IEEE, 2020.

R. Grosse, J. Bae, C. Anil, N. Elhage, A. Tamkin, A. Tajdini, B. Steiner, D. Li, E. Durmus, E. Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

K. Guu, A. Webson, E. Pavlick, L. Dixon, I. Tenney, and T. Bolukbasi. Simfluence: Modeling the influence of individual training examples by simulating training runs. *arXiv preprint arXiv:2303.08114*, 2023.

N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

K. Hanawa, S. Yokoi, S. Hara, and K. Inui. Evaluation of similarity-based explanations. *arXiv preprint arXiv:2006.04528*, 2020.

Y. Hao, Y. Cao, and L. Mou. Flora: Low-rank adapters are secretly gradient compressors. In *International Conference on Machine Learning*, pages 17554–17571. PMLR, 2024.

S. Hara, A. Nitanda, and T. Maehara. Data cleansing for models trained with sgd. *Advances in Neural Information Processing Systems*, 32, 2019.

E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.

P. Hu, J. Melkonian, W. Tang, H. Zhao, and J. W. Ma. Grass: Scalable data attribution with gradient sparsification and sparse projection. *arXiv preprint arXiv:2505.18976*, 2025.

A. Ilyas, S. M. Park, L. Engstrom, G. Leclerc, and A. Madry. Datamodels: Predicting predictions from training data. *arXiv preprint arXiv:2202.00622*, 2022.

R. Karakida, S. Akaho, and S.-i. Amari. Pathological spectra of the fisher information metric and its variants in deep neural networks. *arXiv preprint arXiv:1910.05992*, 2019.

P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pages 1885–1894. PMLR, 2017.

Y. Kwon, E. Wu, K. Wu, and J. Y. Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *International Conference on Learning Representations*, volume 2024, pages 21921–21942, 2024.

S. Li, H. Le, J. Xu, and M. Salzmann. Learning to weight parameters for data attribution. *arXiv preprint arXiv:2506.05647*, 2025.

K. Liang, B. Liu, L. Chen, and Q. Liu. Memory-efficient llm training with online subspace descent. *Advances in Neural Information Processing Systems*, 37:64412–64432, 2024.

S.-Y. Liu, C.-Y. Wang, H. Yin, P. Molchanov, Y.-C. F. Wang, K.-T. Cheng, and M.-H. Chen. Dora: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024.

B. Mlodozeniec, I. Reid, S. Power, D. Krueger, M. Erdogdu, R. Turner, and R. Grosse. Distributional training data attribution: What do influence functions sample? *Advances in Neural Information Processing Systems*, 38:10057–10091, 2026.

T. Olmo, A. Ettinger, A. Bertsch, B. Kuehl, D. Graham, D. Heineman, D. Groeneveld, F. Brahman, F. Timbers, H. Ivison, et al. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

S. M. Park, K. Georgiev, A. Ilyas, G. Leclerc, and A. Madry. Trak: Attributing model behavior at scale. *arXiv preprint arXiv:2303.14186*, 2023.

G. Pruthi, F. Liu, S. Kale, and M. Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33:19920–19930, 2020.

A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

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

A. Schioppa. Efficient sketches for training data attribution and studying the loss landscape. *Advances in Neural Information Processing Systems*, 37:37692–37735, 2024.

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

S. P. Singh, G. Bachmann, and T. Hofmann. Analytic insights into structure and rank of neural network hessian maps. *Advances in Neural Information Processing Systems*, 34:23914–23927, 2021.

S. Teso, A. Bontempelli, F. Giunchiglia, and A. Passerini. Interactive label cleaning with example-based explanations. *Advances in Neural Information Processing Systems*, 34:12966–12977, 2021.

T. Vogels, S. P. Karimireddy, and M. Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. *Advances in Neural Information Processing Systems*, 32, 2019.

T. Vogels, S. P. Karimireddy, and M. Jaggi. Practical low-rank communication compression in decentralized deep learning. *Advances in Neural Information Processing Systems*, 33:14171–14181, 2020.

A. Wang, E. Nguyen, R. Yang, J. Bae, S. A. McIlraith, and R. Grosse. Better training data attribution via better inverse hessian-vector products. *arXiv preprint arXiv:2507.14740*, 2025a.

B. Wang, C. Lee, N. Lee, S.-C. Lin, W. Dai, Y. Chen, Y. Chen, Z. Yang, Z. Liu, M. Shoeybi, B. Catanzaro, and W. Ping. Nemotron-cascade: Scaling cascaded reinforcement learning for general-purpose reasoning models. 2025b.

H. Wang, S. Sievert, S. Liu, Z. Charles, D. Papailiopoulos, and S. Wright. Atomo: Communication-efficient learning via atomic sparsification. *Advances in neural information processing systems*, 31, 2018.

J. T. Wang, D. Song, J. Y. Zou, P. Mittal, and R. Jia. Capturing the temporal dependence of training data influence. In *International conference on learning representations*, volume 2025, pages 11236–11273, 2025c.

M. Wojnowicz, B. Cruz, X. Zhao, B. Wallace, M. Wolff, J. Luan, and C. Crable. "influence sketching": Finding influential samples in large-scale regressions. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 3601–3612. IEEE, 2016.

Y. Wu, X. Zhu, C. Wu, A. Wang, and R. Ge. Dissecting hessian: Understanding common structure of hessian in neural networks. *arXiv preprint arXiv:2010.04261*, 2020.

C.-K. Yeh, J. Kim, I. E.-H. Yen, and P. K. Ravikumar. Representer point selection for explaining deep neural networks. *Advances in neural information processing systems*, 31, 2018.

J. Zhao, Z. Zhang, B. Chen, Z. Wang, A. Anandkumar, and Y. Tian. Galore: Memory-efficient llm training by gradient low-rank projection. In *International Conference on Machine Learning*, pages 61121–61143. PMLR, 2024.

A Positioning LoRIF within the TDA Landscape

TDA methods differ not only in computational cost, but also in the attribution quantity they estimate. Some methods estimate global data value, some estimate query-specific influence, some model counterfactual training behavior, and others retrieve examples by representation or text similarity. Table 4 summarizes these differences in terms of the information each method family uses and the artifacts required to answer attribution queries.

Subset valuation and simulation. Subset-based valuation methods estimate the effect of data by comparing model behavior across different training subsets. Data Shapley [Ghorbani and Zou, 2019] defines data value through a Shapley-value objective and approximates it with Monte Carlo and related estimators. Datamodeling [Ilyas et al., 2022] learns predictors of model behavior from subset membership across many trained models. Simfluence [Guu et al., 2023] instead learns a simulator of training loss trajectories under hypothetical curricula. These methods are closely tied to counterfactual training behavior, but their required training or simulator-construction cost differs from LoRIF's final-checkpoint attribution setting.

Trajectory-aware attribution. Trajectory-aware methods account for the optimization path rather than only the final model. TracIn [Pruthi et al., 2020] aggregates gradient dot products across saved checkpoints. SGD-Influence [Hara et al., 2019] and HyDRA [Chen et al., 2021] estimate data effects by propagating influence or hypergradients through training dynamics. SOURCE [Bae et al., 2024] connects implicit-differentiation and unrolling-based approaches using an approximate unrolled formulation. To reduce the cost of recording gradients along the training trajectory, Wang et al. [2025c] combine exact activation/backpropagation factorization with two-sided projection. For sequence data, this exact factorization retains the token dimension; LoRIF instead compresses the token-aggregated projected gradient to rank c. These trajectory-aware approaches can capture effects missed by final-checkpoint approximations, but require access to the training trajectory or an approximation of it.

Representation and text similarity. Representation-based methods retrieve examples using model activations, learned embeddings, or text-level similarity. Representer point selection [Yeh et al., 2018] decomposes predictions into contributions from training-point representations with learned representer values. RepSim-style methods [Hanawa et al., 2020] compare training and query representations directly, and text-retrieval baselines use lexical or embedding similarity. These methods are often efficient and useful retrieval baselines, but they do not explicitly estimate curvature-aware parameter influence.

Gradient similarity and parameter-space influence. Gradient-similarity methods score training examples using dot products or cosine similarities between training and query gradients, often corresponding to an identity-curvature approximation. Classical influence-function methods instead precondition gradients with an inverse-Hessian-vector product or curvature approximation. These approaches estimate model-dependent influence, but full-parameter gradients and curvature solvers can be expensive for large language models, especially when many query-time attributions are required.

Projection-based influence functions and LoRIF. Projection-based influence methods reduce the dimensionality of per-example gradients before storing or scoring them, making final-checkpoint influence attribution more practical at scale. This is the closest method family to LoRIF: training gradients are indexed once, and many query-specific attributions are answered from the stored projected gradients. LoRIF keeps this setting but replaces full projected-gradient matrices with low-rank factors and replaces dense curvature storage with a truncated-SVD approximation.

B Additional Details on Experimental Setup

B.1 Hardware

All experiments were conducted on an x86_64 server with the following specifications:

- **CPU**: 2× AMD EPYC 9454 (48 cores/socket, 2 threads/core; 96 cores / 192 threads total)

Table 4: **A coarse taxonomy of training-data attribution methods.** The table summarizes information requirements and typical stored representations.

Family / method	What the score uses	Extra training information	Per-example artifact stored for repeated queries
Subset-based valuation / simulation	Marginal effects of including or excluding training subsets; sometimes a learned simulator of model behavior from subset membership.	Usually many subset-trained models or evaluations; in-run variants instead require logging during the training run.	Usually no query-specific gradient store; may store global values or a learned simulator.
Trajectory-aware attribution	Gradients or parameter changes accumulated along the optimization path.	Saved checkpoints, update history, or an unrolled/differentiable training trajectory; no new retraining if the trajectory was stored.	Sometimes checkpoint-gradient caches; not a final-checkpoint-only representation.
Representation or text similarity	Text overlap, embedding similarity, or hidden-state similarity between training and query examples.	No retraining or saved training trajectory.	Text, embedding, or representation store for nearest-neighbor retrieval.
Gradient similarity / Hessian-free IF	Dot products between training and query gradients, often using selected layers or an identity curvature approximation.	No retraining or saved training trajectory.	Optional full- or layer-gradient cache; otherwise training gradients must be streamed or recomputed.
Parameter-space IF / iHVP	Training and query gradients preconditioned by an inverse-Hessian-vector product or curvature approximation.	No retraining or saved training trajectory, but requires an iHVP or curvature solver.	Optional full- or layer-gradient cache; many implementations stream or recompute training gradients.
Projection-based IF	Projected per-example gradients, optionally with curvature or preconditioning.	No saved training trajectory; typically computed from one or a few final checkpoints.	Full projected-gradient store for all training examples.
LoRIF (this work)	IF-style scores using low-rank projected-gradient factors and a truncated-SVD curvature approximation.	No saved training trajectory; same final-checkpoint setting as projection-based IF methods.	Rank-c projected-gradient factors

- **Storage**: NVMe SSD (1× 7 TB Samsung MZWLO7T6HBLA-00A07 + 1× 894 GB M.2 NVMe RAID kit)
- **GPU**: 1× NVIDIA H200 with 141 GiB memory, unless specified otherwise.

B.2 Hyperparameters

We report the values of f, c and r used in our experiments in Tables 5, 6, 7. Other hyperparameters are reported below.

- **Effective projection dimension** D: This is controlled by the choice of f via $D = d_1 \times d_2 = \frac{I}{f} \times \frac{O}{f}$ for each layer, where I and O are the input and output dimensions of the layer. For GPT2-small, the largest $I \cdot O$ is up to $3072 \times 768 \approx 2.3 \times 10^6$ per layer. For OLMo3-7B, the largest $I \cdot O$ is up to $11008 \times 4096 \approx 4.5 \times 10^7$ per layer. For Apertus-70B, the largest $I \cdot O$ is up to $43008 \times 8192 \approx 3.5 \times 10^8$ per layer.
- **Damping** λ: Grosse et al. [2023] and Choe et al. [2024] recommend a damping term $\lambda = 0.1 \times \text{mean}(\Lambda) = 0.1 \times \text{tr}((\tilde{G}_i^\ell)^\top \tilde{G}_i^\ell)/D$, where Λ is all the eigenvalues of the Hessian H. We follow this recommendation and set the damping term for each layer individually for

all LoGRA experiments. For LoRIF experiments, we empirically found that replacing Λ with the top-r eigenvalues works well. [1]

- **Number of power iterations for rank factorization**: 8 when $c = 1$ and 16 when $c > 1$ for all experiments on all models. Further increasing the number does not improve the attribution quality.
- **Number of power iterations for truncated SVD**: 3 for all experiments on all models. Further increasing the number does not improve the attribution quality.
- **Training hyperparameters**: For GPT2-small, we follow the exactly same training configurations in Choe et al. [2024]. For OLMo3-7B and Apertus-70B, we use the public checkpoints released by the developers and never trained them on our own due to resource constraints.

B.3 Baseline Methods.

Our primary comparisons are against influence-function-style methods that share LoRIF's final-checkpoint attribution setting

- **LoGRA** [Choe et al., 2024]: A projection-based method that stores projected per-example gradients and uses a damped Gauss-Newton Hessian approximation. Compared to TRAK [Park et al., 2023], LoGRA is more scalable as it leverages two-sided random projections.
- **TrackStar** [Chang et al., 2024]: Similar to LoGRA, with innovations on hessian approximation, normalization, etc., aiming to improve fact tracing tasks specifically.
- **EK-FAC** [Grosse et al., 2023]: A parameter-space method that leverages the eigenvalue-corrected K-FAC Hessian approximation for faster iHVP computation, requiring recomputation of training gradients for each query batch.
- **GradDot** [Charpiat et al., 2019, Pruthi et al., 2020]: A simple gradient-based method that uses a dot product between the query and training gradients to approximate the influence. Two-sided random projections are applied to the gradients.
- **RepSim** [Hanawa et al., 2020]: A simple method that computes the cosine similarity between the representation of the query and the training examples. We use the hidden states of the last token in the last layer of the model as the representation.

B.4 Models and Datasets.

We evaluate LoRIF using the following models and datasets of different scales.

- **GPT2-small** [Radford et al., 2019]: A 124M-parameter language model pretrained on the WikiText-103 dataset. Texts in the training split of the dataset are concatenated and divided into 232,585 sequences of 512 tokens. We use the validation split of the dataset for evaluation, which contains 487 sequences of the same length.
- **OLMo3-7B** [Olmo et al., 2025]: A 7B-parameter language model with open-source checkpoints and datasets. Instead of training the model ourselves, we adopt the "Thinking SFT" checkpoint,, which was finetuned on the Dolci Think SFT dataset, containing 2.2M training examples. Each example is applied the default chat template and truncated to 1024 tokens. To build a dataset for evaluation, we randomly sampled 1,000 questions from the Nemotron Cascade SFT dataset [Wang et al., 2025b] and use the model to generate the responses. Only "assistant" tokens are used for gradient computation.
- **Apertus-70B** [Apertus et al., 2025]: A 70B-parameter language model with open-source model checkpoints and datasets. Similarly to OLMo3, we adopt its "Instruct" model,, which was fine-tuned on the publicly available SFT dataset, containing 3.8M examples, each truncated to 512 tokens. We sampled 1,000 questions from the Nemotron Cascade SFT dataset and generate the responses using the model.

[1] In our public code release, we instead choose $\lambda = 0.5 \times \mathrm{tr}((\tilde{G}_i^\ell)^\top \tilde{G}_i^\ell)/D$ for LoRIF because we found it to be more stable across different choices of r. We also implemented a new "tail correction" configuration that uses an isotropic approximation of the discarded curvature spectrum. Check `https://github.com/doub7e/LoRIF` for details.

B.5 Evaluation Metrics

Storage. Total space used for storing the per-example training gradients or the low-rank factors. We do not consider the storage costss of H^{-1} or V_r because they do not scale linearly with the number of training examples.

Query latency : Wall-clock time for computing the the influence scores for a batch of query examples, including the time for loading all training examples' gradients or low-rank factors. The main comparison tables use 487 queries for GPT2-small and 1,000 for OLMo-3-7B and Apertus-70B. The separate timing breakdown in Figure 3 uses its stated $N_q = 1{,}000$ setting.

Linear Datamodeling Score (LDS). LDS [Park et al., 2023] evaluates whether an attribution method τ can predict how model outputs change when trained on different data subsets. The procedure works as follows:

1. **Sample subsets:** Generate M random subsets $\{\mathcal{S}^m\}_{m=1}^{M}$ of the training data, each containing a fraction α of the full dataset.
2. **Compute outputs:** For each query x_{query} and subset $\mathcal{S}^m$:
   - *Actual output:* Retrain a model θ_m on $\mathcal{S}^m$ and compute $y_m^{\text{actual}} = f(x_{\text{query}}; \theta_m)$.
   - *Predicted output:* Sum the attribution scores: $y_m^{\text{predicted}} = \sum_{x^{(i)} \in \mathcal{S}^m} \tau(x_{\text{query}}, \mathcal{S})^{(i)}$.
3. **Compute correlation:** $\text{LDS}(\tau, x_{\text{query}}) = \rho\big(\{y_m^{\text{actual}}\}_{m=1}^{M}, \{y_m^{\text{predicted}}\}_{m=1}^{M}\big)$, where ρ is the Spearman rank correlation.

Higher LDS (closer to 1) indicates better agreement with subset-retraining outputs under the additive prediction used by the metric.

Our implementation: We use $\alpha = 0.5$ (50% data) and $M = 100$ subsets. For each subset, we train 5 GPT2-small models from scratch and average their outputs to reduce variance. The output function is the negative token-averaged cross-entropy loss, $f(x_{\text{query}}; \theta_m) = -\ell(x_{\text{query}}; \theta_m)$. We correlate this utility with the sum of proponent scores over the examples retained in each subset, so that larger predicted values correspond to lower query loss.

Tail-patch score. Tail-patch score [Chang et al., 2024] is a scalable alternative to LDS that requires no retraining. For each query, the procedure identifies the top-k training proponents according to the attribution method, performs one gradient step (a "tail patch") on the final checkpoint using these k examples, and measures the increase in the query's target sequence probability. Higher scores indicate better retrieval of genuinely influential training data.

Our implementation: We deviate from Chang et al. [2024] in two ways:

- **Batched evaluation:** Instead of performing k separate tail-patches (one per retrieved example) and averaging, we follow Li et al. [2025] and use the top-k examples as a single batch for one tail-patch. This is substantially more efficient, which is important when evaluating large models.
- **Adapted hyperparameters:** Chang et al. [2024] recommend using the original training hyperparameters, but this is infeasible for models trained on thousands of GPUs with very large batch sizes. We manually tune the learning rate and batch size to ensure meaningful parameter updates without model collapse. For OLMo3-7B: $\text{lr} = 10^{-5}$, $k = 8$. For Apertus-70B: $\text{lr} = 3 \times 10^{-6}$, $k = 48$ (micro-batch size 3, 16 gradient accumulation steps, 4× H200 tensor parallelism).

Alignment between LDS and tail-patch. Figure 5 reveals a strong linear alignment between LDS and tail-patch score across attribution methods (except RepSim). This confirms that methods which accurately predict retraining outcomes (LDS) also retrieve influential top-k training examples (tail-patch). Consequently, tail-patch score can serve as a faithful metric for evaluating attribution quality on frontier models where LDS is intractable.

Two caveats are worth noting. First, RepSim deviates most from the trend line; we suspect this is because representation similarity is not gradient-based, and thus does not identify examples that

would yield the largest parameter updates in a tail-patch step. Second, the two metrics measure different aspects: LDS aggregates influence scores over *all* training examples to predict subset-level effects, whereas tail-patch evaluates only the *top-k* retrieved examples. Despite this difference, the strong correlation suggests that methods ranking examples well globally also rank the top-k well.

C Preprocessing and Query-Time Costs

Preprocessing consists of two stages:

1. **Stage 1: Gradient computation and storage.** Compute and save the (projected) per-example gradients for all training examples. For LoRIF, this includes solving the rank-c factorization via power iteration. For EK-FAC, this stage instead computes and saves the covariances and eigenvalue corrections per layer (see [Grosse et al., 2023] for details); EK-FAC does not store full gradients.
2. **Stage 2: Inverse Hessian approximation.** For LoGRA, form and store $(G^\top G + \lambda I)^{-1}$ per layer. For LoRIF, perform randomized SVD to obtain V_r and Σ_r, then store them.

Tables 5, 6, and 7 report preprocessing times for each model. Runtime depends on many uncontrolled factors and can vary; we report representative measurements. We make the following observations:

- **Stage 1:** For LoRIF with $c = 1$, solving the rank factorization via power iteration adds negligible time. When $c \geq 4$, factorization becomes more expensive due to additional iterations and column orthonormalization. As f decreases (i.e., D increases), gradient projection and disk I/O cause noticeable overhead, especially for LoGRA, which writes significantly more data. EK-FAC instead computes layerwise covariance and eigenvalue-correction statistics in this stage; its query-time gradient recomputation is accounted for separately. Frequent data transfers between CPU and GPU also contribute to the long processing time for EK-FAC.
- **Stage 2:** Heavily dependent on f. Smaller f (larger D) leads to longer Stage 2 time. When c and r are large, LoRIF's Stage 2 can become slower than LoGRA's (see the complexity analysis in Section 3.3); this occurs on GPT2 when f is small and D large.
- **Large-scale models:** For OLMo3-7B and Apertus-70B, LoRIF, which uses smaller f and requires power iterations, adds only modest preprocessing overhead compared to LoGRA.

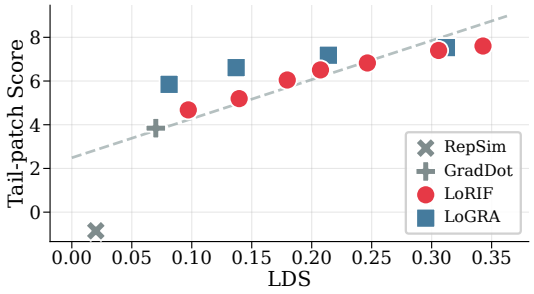


Figure 5: **LDS vs. tail-patch score** on GPT2-small. Each point is a method–configuration pair. The dashed line is a linear fit. RepSim (non-gradient-based) deviates most from the trend.

Table 5: **Preprocessing time on GPT2-small** (233k training examples). EK-FAC has only Stage 1 (no stored gradients). "–" indicates not applicable.

Method	f	c	r	**Stage 1**	**Stage 2**	**Total**
EK-FAC	–	–	–	7 hr	–	7 hr
LoGRA	64	–	–	15 min	1 s	15 min
LoGRA	32	–	–	15.5 min	6 s	16 min
LoGRA	16	–	–	18 min	1 min	19 min
LoGRA	8	–	–	23.5 min	13 min	37 min
LoRIF	32	1	2^{10}	16 min	12 s	16 min
LoRIF	16	1	2^{11}	16 min	1.5 min	18 min
LoRIF	8	1	2^{12}	16.5 min	20 min	37 min
LoRIF	4	1	2^{12}	17.5 min	30 min	48 min
LoRIF	4	4	2^{13}	41 min	1 hr	1.7 hr
LoRIF	4	16	2^{14}	50 min	2.5 hr	3.5 hr
LoRIF	4	32	2^{14}	59 min	5 hr	6 hr

Table 6: **Preprocessing time on OLMo3-7B** (2.2M training examples).

Method	f	c	r	**Stage 1**	**Stage 2**	**Total**
LoGRA	360	–	–	68 hr	2 min	68 hr
LoGRA	256	–	–	67 hr	4 min	67 hr
LoGRA	180	–	–	68 hr	6.5 min	68 hr
LoGRA	128	–	–	67.5 hr	13.5 min	68 hr
LoRIF	128	1	2^8	67.5 hr	2 min	68 hr
LoRIF	64	1	2^8	68 hr	9 min	68 hr
LoRIF	32	1	2^8	68 hr	30 min	69 hr
LoRIF	16	1	2^8	70 hr	2 hr	72 hr

Table 7: **Preprocessing time on Apertus-70B** (3.8M training examples). To be able to compute gradients for this large model, we used two H200 GPUs and applied tensor parallelism.

Method	f	c	r	**Stage 1**	**Stage 2**	**Total**
LoGRA	512	–	–	180 hr	18 min	180 hr
LoRIF	256	1	2^9	185 hr	30 min	186 hr
LoRIF	64	1	2^9	195 hr	5 hr	200 hr

Table 8: **Preprocessing/query-time break-even.** Time pairs are LoGRA / LoRIF. $K^\star$ counts query batches, not individual queries. The first whole batch reaching the threshold is $\lceil K^\star \rceil$.

Setting	B	Preprocessing	Latency/batch	$K^\star$
GPT2-small, low storage	487	16 / 18 min	16.4 / 2.93 s	8.9
OLMo-3-7B, higher quality	1,000	68 / 72 hr	782.1 / 469.7 s	46.1
Apertus-70B, efficient	1,000	180 / 186 hr	1087.9 / 222.5 s	25.0
Apertus-70B, higher quality	1,000	180 / 200 hr	1087.9 / 843.8 s	295.0

C.1 Amortization across Query Batches

Preprocessing is an offline cost incurred once for a fixed checkpoint and attribution corpus. Query latency is incurred again whenever a new batch is scored against the stored index. We report both because a lower cumulative runtime and a shorter wait for each new query are different practical objectives. Let P_m be method m's preprocessing time and $L_m(B)$ its measured latency for a batch of B queries. After K such batches, the cumulative time is

$$C_m(K) = P_m + KL_m(B). \tag{10}$$

When LoRIF has higher preprocessing cost but lower query latency, the break-even point is

$$K^\star = \frac{P_{\mathrm{LoRIF}} - P_{\mathrm{LoGRA}}}{L_{\mathrm{LoGRA}}(B) - L_{\mathrm{LoRIF}}(B)}, \qquad Q^\star = BK^\star. \tag{11}$$

Here $B = 487$ for the GPT2-small main comparison and $B = 1{,}000$ for OLMo-3-7B and Apertus-70B. Table 8 uses the rounded preprocessing totals in Tables 5–7; the resulting thresholds are approximate. The GPT2-small row compares LoGRA $f = 32$ with LoRIF $f = 16, c = 1, r = 2^{11}$. The OLMo-3-7B row compares LoGRA $f = 128$ with LoRIF $f = 16, c = 1, r = 2^8$. The Apertus-70B rows compare LoGRA $f = 512$ with LoRIF $f = 256$ or 64, respectively, both with $c = 1, r = 2^9$. These comparisons have similar or higher measured attribution quality for LoRIF. The additional preprocessing is amortized within tens of query batches for several configurations, while the higher-quality Apertus configuration takes substantially longer. Not every configuration has a wall-clock break-even: GPT2-small with $f = 4, c = 32$ improves LDS and reduces storage relative to LoGRA $f = 8$, but has higher query latency and preprocessing cost. For an already constructed index, query latency remains relevant independently of amortization, since it determines the delay in inspecting each newly submitted query.

D Ablation: Separating the Two Low-Rank Components

LoRIF uses two low-rank components: rank-c factorization of projected per-example gradients and truncated-SVD curvature approximation. Table 9 ablates each component on GPT2-small. The two components address different bottlenecks. Removing truncated SVD keeps the stored gradients compact, but leaves the curvature bottleneck; as a result, the method becomes out-of-memory in the high-dimensional settings that give the best LDS. Removing rank factorization preserves the curvature approximation and can achieve strong LDS, but it restores the projected-gradient storage and I/O bottleneck. Thus, both components are needed to make high-dimensional projected influence practical.

E Approximation Diagnostics and Hyperparameter Choice

LoRIF introduces two approximations: rank-c factorization of projected per-example gradients and rank-r truncated SVD for the curvature approximation. This appendix studies both approximations and explains how we choose the hyperparameters (f, c, r) in practice.

E.1 Rank-c Factorization Error

For each model, we collect 1,000 per-sample projected gradients and compute the relative Frobenius reconstruction error of their rank-c approximations. Table 10 reports results at comparable projected

Table 9: **Ablation of LoRIF components on GPT2-small.** Rank factorization reduces persistent gradient storage and query-time I/O, while truncated SVD removes the curvature-memory bottleneck. Storage denotes the persistent training-gradient storage.

Method	f	c	r	LDS ↑	Storage	Storage/Ex.	Latency
LoRIF w/o truncated SVD	8	1	–	0.1868 ± 0.013	8.3 GiB	0.04 MiB	72.7 s
LoRIF w/o truncated SVD	4	16/32	–	OOM	–	–	–
LoRIF w/o rank factorization	4	–	2^{14}	$\mathbf{0.3876 \pm 0.0116}$	2297.7 GiB	10.1 MiB	1548.0 s
LoRIF	4	16	2^{14}	0.3058 ± 0.013	255.5 GiB	1.12 MiB	165.9 s
LoRIF	4	32	2^{14}	0.3428 ± 0.012	511.0 GiB	2.25 MiB	287.1 s
LoRIF w/o truncated SVD	4	1	–	OOM	–	–	–
LoRIF w/o rank factorization	4	–	2^{12}	$\mathbf{0.3525 \pm 0.0118}$	2297.7 GiB	10.1 MiB	931.6 s
LoRIF	4	1	2^{12}	0.2073 ± 0.013	16.6 GiB	0.07 MiB	25.5 s
LoRIF w/o truncated SVD	16	1	–	0.1414 ± 0.012	4.1 GiB	0.02 MiB	6.57 s
LoRIF w/o rank factorization	16	–	2^{11}	$\mathbf{0.2080 \pm 0.0109}$	143.7 GiB	0.63 MiB	66.9 s
LoRIF	16	1	2^{11}	0.1392 ± 0.011	4.1 GiB	0.02 MiB	2.93 s

Table 10: **Rank-c factorization error of projected per-example gradients.** Errors are relative Frobenius reconstruction errors. EVR denotes the fraction of Frobenius energy captured by the rank-c approximation.

Model	Module	$c = 1$ Err.	$c = 1$ EVR	$c = 4$ Err.	$c = 4$ EVR
GPT2-small (124M)	attn	0.787	37.6%	0.557	67.8%
GPT2-small (124M)	mlp	0.834	30.2%	0.711	49.3%
OLMo-3-7B	attn	0.773	38.8%	0.594	63.5%
OLMo-3-7B	mlp	0.827	31.2%	0.694	51.3%
Apertus-70B	attn	0.479	74.3%	0.211	93.8%
Apertus-70B	mlp	0.641	55.0%	0.454	77.1%

dimensions, grouped by module type. EVR denotes the fraction of Frobenius energy captured by the rank-c approximation.

Two observations are important. First, the rank-c approximation does not degrade when moving from GPT2-small to OLMo-3-7B: the two models have similar reconstruction errors at matched projected dimensions. Second, Apertus-70B is substantially more compressible in our measurements, especially in attention modules. We do not interpret this as a pure model-size effect, since architecture, data, and projected dimensions also differ. The result nevertheless shows that the rank-c approximation remains well controlled at larger model scales.

The compressibility is consistent with the structure of transformer linear-layer gradients. For a sample i, let

$$A_i = X_i P_{\text{in}} \in \mathbb{R}^{T \times d_1}, \qquad B_i = \delta Y_i P_{\text{out}} \in \mathbb{R}^{T \times d_2},$$

so the projected per-example gradient is

$$\widetilde{G}_i = B_i^\top A_i = \sum_{t=1}^{T} B_{i,t}^\top A_{i,t}.$$

Thus, $\widetilde{G}_i$ is a sum of token-level outer products. Although its exact rank can be as large as the sequence length T, the token-level terms are highly correlated in practice, so much of the Frobenius energy concentrates in a small number of directions. LoRIF exploits this by replacing the exact token-level factorization with a much smaller rank-c approximation of the already aggregated projected gradient matrix.

E.2 Spectral Concentration and Choosing the Truncation Rank r

We next analyze the aggregate projected-gradient matrix used in the curvature approximation. For a layer, let $G \in \mathbb{R}^{N \times D_\ell}$ denote the matrix whose rows are projected training gradients, and let

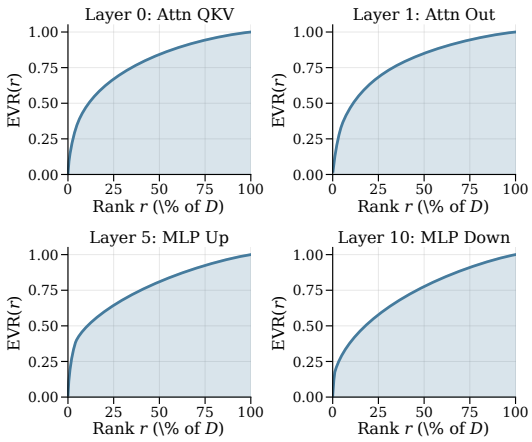


Figure 6: **Spectral concentration of the projected gradient matrix G on GPT2-small.** The curve shows the cumulative explained variance ratio $\mathrm{EVR}(r)$ from Equation (12).

$G = U\Sigma V^\top$ be its SVD. We measure spectral concentration using the cumulative explained-variance ratio

$$\mathrm{EVR}(r) = \frac{\sum_{i=1}^{r} \sigma_i^2}{\sum_{i=1}^{\min(N, D_\ell)} \sigma_i^2}. \tag{12}$$

Figure 6 plots $\mathrm{EVR}(r)$ for GPT2-small. The GPT2-small spectrum shows moderate concentration. To compare across model scales, Table 11 reports EVR at several truncation ratios using 100K training samples per model. The projection factors are chosen to give comparable effective projected dimensions within each module type.

Table 11: **Spectral concentration of projected training-gradient matrices.** EVR@$p\%$ denotes the explained variance captured by the top $p\%$ singular directions.

Model	**Module**	D	**EVR@10%**	**EVR@25%**	**EVR@50%**
GPT2-small, $f = 32$	attn	1152	0.49	0.67	0.83
GPT2-small, $f = 32$	mlp	2304	0.42	0.60	0.79
OLMo-3-7B, $f = 128$	attn	1024	0.43	0.59	0.77
OLMo-3-7B, $f = 128$	mlp	2752	0.36	0.52	0.72
Apertus-70B, $f = 256$	attn	576	0.77	0.84	0.91
Apertus-70B, $f = 256$	mlp	5376	0.53	0.66	0.81

GPT2-small and OLMo-3-7B show similar spectral concentration, indicating that scaling from 124M to 7B parameters does not meaningfully degrade the truncated-SVD approximation in our measurements. Apertus-70B shows stronger concentration, particularly in attention modules. Again, we do not claim that this is solely due to model size; the result should be read as evidence that the approximation remains effective across the model scales and architectures considered here.

Why small r can preserve attribution quality. Moderate EVR can still be sufficient because matrix reconstruction error is not equivalent to attribution error. Let the k-th training gradient be represented as a column vector

$$g_k = G^\top e_k = V\Sigma U^\top e_k = \sum_i \sigma_i U_{k,i} v_i. \tag{13}$$

Therefore, the coordinate of g_k along the right singular vector v_i is

$$g'_{k,i} := v_i^\top g_k = \sigma_i U_{k,i}.$$

For a test gradient g_{te}, define $g'_{te,i} := v_i^\top g_{te}$, and collect the retained coordinates as $g'_{te} = V_r^\top g_{te}$ and $g'_k = V_r^\top g_k$

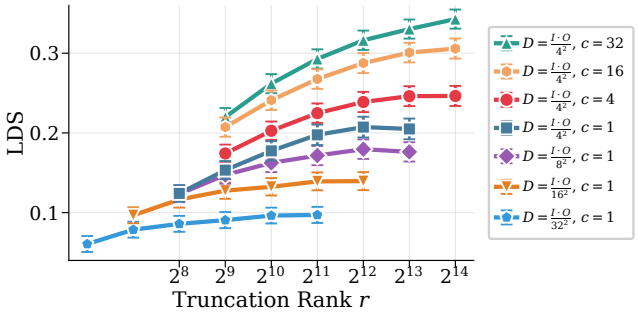


Figure 7: **LDS vs. truncation rank r with rank-c factorization.** Different curves correspond to different projection dimensions D and factorization ranks c. LDS saturates at $r \ll D$, especially for small c, confirming that truncated SVD remains effective when combined with low-rank gradient storage. Experiments are run on GPT2-small.

In Eq. (9), the curvature-correction term decomposes over singular directions as

$$\frac{1}{\lambda^2}(g'_{te})^\top \left(\Sigma_r^{-2} + \frac{1}{\lambda} I_r\right)^{-1} g'_k = \sum_{i=1}^{r} w_i \, g'_{te,i} g'_{k,i}, \tag{14}$$

where

$$w_i = \frac{1}{\lambda^2}\left(\sigma_i^{-2} + \frac{1}{\lambda}\right)^{-1} = \frac{\sigma_i^2}{\lambda(\lambda + \sigma_i^2)}. \tag{15}$$

The coefficient w_i is monotone in σ_i^2 and suppresses directions with small singular values. When $\sigma_i^2 \ll \lambda$, we have $w_i \approx \sigma_i^2/\lambda^2$. Moreover, training gradients themselves carry little energy in low-singular-value directions:

$$\frac{1}{N}\sum_{k=1}^{N}(g'_{k,i})^2 = \frac{1}{N}\sum_{k=1}^{N}\sigma_i^2 U_{k,i}^2 = \frac{\sigma_i^2}{N}.$$

Thus, tail directions are suppressed twice: the inverse-Hessian correction gives them small weight, and training gradients have little average energy in those directions.

This argument can also be stated as a simple worst-case bound. Since $|g'_{k,i}| = |\sigma_i U_{k,i}| \le \sigma_i$ and $|g'_{te,i}| \le \|g_{te}\|$, the magnitude of the i-th contribution satisfies

$$|w_i g'_{te,i} g'_{k,i}| \le \frac{\sigma_i^2}{\lambda(\lambda + \sigma_i^2)} \, \sigma_i \, \|g_{te}\|.$$

For $\sigma_i^2 \ll \lambda$, this bound scales as $O(\sigma_i^3/\lambda^2)$. The bound is loose, but it shows that tail directions decay rapidly even without assuming that the test gradient has the same spectral profile as training gradients.

Empirically, LDS saturates well before r reaches D_ℓ. Figure 7 shows that this remains true when truncated SVD is combined with rank-c gradient factorization.

E.3 Why Projection Precedes Rank Factorization

LoRIF first applies LoGRA-style two-sided projection and then factorizes the projected per-example gradient. These two steps solve different problems. Projection maps each $O \times I$ gradient matrix to a $d_2 \times d_1$ matrix, making the curvature approximation tractable. Rank-c factorization then reduces persistent storage and query-time I/O within this projected space.

Without projection, the curvature stage would still operate in the original dimension IO, which can be prohibitively large for modern transformer layers. Thus, factorization is not a substitute for projection; it is an additional compression stage applied after projection.

E.4 Choosing f, c, and r

The three hyperparameters control different parts of the approximation. The projection factor f controls the effective projected dimension

$$D_\ell = d_1 d_2 = \frac{IO}{f^2},$$

so smaller f means larger projected dimension. Increasing D_ℓ expands the attribution subspace and usually improves attribution quality, but increases storage, I/O, and curvature cost. The factorization rank c controls how accurately each projected per-example gradient matrix is reconstructed within the chosen projected space. The truncation rank r controls how much curvature information is retained.

We use the following empirical selection rule:

1. Choose the smallest feasible projection factor f, equivalently the largest feasible projected dimension D_ℓ.
2. Keep c as small as possible, increasing it only when quality remains insufficient and further decreasing f is infeasible.
3. Choose r as large as the memory budget allows, until attribution quality saturates.

This rule reflects the empirical trend that increasing D_ℓ is often more valuable than increasing c under a fixed budget. Increasing D_ℓ expands the space in which attribution is computed, whereas increasing c only improves reconstruction within an already chosen projected space. Higher c is therefore most useful in high-quality regimes where D_ℓ has already been pushed as far as possible. For example, on GPT2-small at $f = 4$, increasing c from 1 to 16 or 32 substantially improves LDS, but also increases storage. On OLMo-3-7B and Apertus-70B, we use $c = 1$ because increasing D_ℓ gives a better quality–efficiency tradeoff in the evaluated regimes.

The choice of r follows the same budget-driven principle. Larger r retains more curvature directions, but the benefit saturates because small-singular-value directions are suppressed by the inverse-Hessian weighting described above. We use this rule as an empirical budget-allocation heuristic; the preferred configuration can vary with the model, data, and attribution metric.

F Comparison and Combination with GraSS/FactGraSS

F.1 Projection Construction and Persistent Storage

GraSS/FactGraSS [Hu et al., 2025] and LoRIF improve different stages of projected-gradient attribution. For a linear layer, FactGraSS applies random or learned masks to the input activations and backpropagated gradients, constructs a smaller intermediate gradient matrix, and applies a sparse Johnson–Lindenstrauss transform (SJLT). Its output is a dense D-dimensional vector per example. Thus, masking reduces the cost of constructing the projected gradient, while storing and loading the resulting vectors still requires $O(ND)$ values per layer. LoRIF instead stores each projected gradient matrix in rank-c form, requiring $O(Nc(d_1 + d_2))$ values per layer, and stores the curvature representation in $O(Dr)$ space. The tradeoff between D and c concerns the representation of an already projected gradient under a storage budget; it does not require a particular projection algorithm. This distinction motivates both a matched-storage comparison and an experiment combining masking with factorized storage.

F.2 Matched-Storage Attribution Quality

We implement random-mask and selective-mask FactGraSS in the GPT2-small setting and compare their LDS with LoRIF at two matched storage budgets. Table 12 reports the projection factor f, factorization rank c, and curvature rank r, where applicable. The $\pm$ values are query-bootstrap confidence-interval half-widths. LoRIF achieves higher LDS than both FactGraSS variants at each tested storage budget. This comparison isolates persistent storage and attribution quality; the next experiment examines masking as a complementary optimization of projection construction.

Table 12: **FactGraSS and LoRIF at matched persistent storage on GPT2-small.** Random-mask FactGraSS is denoted FactGraSS-RM.

Method	f	c	r	Storage	LDS ↑
Medium storage					
FactGraSS-RM	48	–	–	16.6 GiB	0.0972 ± 0.0098
Selective-FactGraSS	48	–	–	16.6 GiB	0.0952 ± 0.0095
LoRIF	4	1	4096	16.6 GiB	$\mathbf{0.2068 \pm 0.0131}$
Low storage					
FactGraSS-RM	96	–	–	4.1 GiB	0.0533 ± 0.0089
Selective-FactGraSS	96	–	–	4.1 GiB	0.0555 ± 0.0093
LoRIF	16	1	2048	4.1 GiB	$\mathbf{0.1388 \pm 0.0111}$

Table 13: **GraSS-style masking combined with LoRIF.** Medium uses $f = 4, c = 1, r = 4096$; Low uses $f = 16, c = 1, r = 2048$. Time is in milliseconds per example for the hook operations specified below.

Regime	Method	Time	Speedup	LDS ↑
Medium	LoRIF	0.5687	1.00×	0.2068 ± 0.0131
	RM-LoRIF	0.4082	1.39×	0.1990 ± 0.0128
	SM-LoRIF	0.4083	1.39×	0.2066 ± 0.0130
Low	LoRIF	0.2805	1.00×	0.1388 ± 0.0111
	RM-LoRIF	0.2080	1.35×	0.1259 ± 0.0117
	SM-LoRIF	0.2082	1.35×	0.1141 ± 0.0115

F.3 Masking within the LoRIF Pipeline

We also apply either a random or learned row/column mask before LoRIF's gradient contraction and rank-c factorization, omitting FactGraSS's final SJLT. These variants, RM-LoRIF and SM-LoRIF, retain LoRIF's low-rank storage and curvature approximation while using GraSS-style masking during projection construction. At Medium storage, SM-LoRIF has nearly unchanged LDS with a $1.39\times$ speedup in the measured operations. At Low storage, both masks reduce LDS, and random masking retains more quality than selective masking. Masking can therefore complement LoRIF, but the resulting quality–time tradeoff depends on the configuration. Timing includes projection/gather operations, gradient contraction, and rank factorization inside the hooks. It excludes model forward/backward computation, storage I/O, selective-mask fitting, and the query latency in Table 1. The reported speedups apply only to these measured operations, not to the full attribution pipeline.

G Direct Retraining-Based Evaluation

We supplement LDS and tail-patch with two removal-and-retraining experiments. Pointwise LOO measures agreement with individual-example deletion effects in a controlled setting. Full-corpus group removal evaluates the utility of the attribution ranking for selecting data to remove.

G.1 Pointwise Leave-One-Out Validation

Data and training. Following the distributional perspective of Mlodozeniec et al. [2026], we average over training seeds to reduce optimization noise. We use a fixed subset of 1,024 WikiText-103 training sequences and all 487 validation blocks as queries. Before computing attribution, we uniformly sample and fix 50 training examples for individual removal. For each of 50 paired seeds, we train one full-data model and 50 models each omitting one selected example, giving $50(1 + 50) = 2{,}550$ runs. Each run uses GPT2-small, three epochs, batch size 8, learning rate 3×10^{-5}, weight decay 0.01, and FP16 with gradient scaling. Dropout is disabled, deterministic algorithms are enabled, and TF32 is disabled. Within each seed, full-data and LOO runs share initialization, epoch permutations, and minibatch partitions. The removed example is deleted from its original minibatch, reducing that batch from 8 to 7 examples without shifting subsequent batches.

Table 14: **Agreement with pointwise LOO retraining on GPT2-small.** Correlations are computed within each query over 50 randomly selected removals and averaged over 487 queries.

Method	Mean Spearman ↑	95% query-bootstrap CI
LoGRA, $f = 8$	0.3648	[0.3529, 0.3767]
LoRIF, $f = 4, c = 32, r = 256$	**0.3966**	[0.3847, 0.4082]

Measured effects and attribution. For query q, removed example i, and seed s, we measure

$$\Delta\ell_{q,i}^{(s)} = \ell_q(\theta_{-i}^{(s)}) - \ell_q(\theta_{\text{full}}^{(s)}), \qquad \overline{\Delta\ell}_{q,i} = \frac{1}{50}\sum_{s=1}^{50}\Delta\ell_{q,i}^{(s)}. \tag{16}$$

We compute each method's influence scores at all 50 full-data checkpoints and average them over seeds as well. The score direction follows Equation (1): a positive score predicts an increase in query loss after removal. LoRIF uses $f = 4, c = 32, r = 256$ and 16 factorization iterations; LoGRA uses $f = 8$ and an exact per-layer dual solve for the damped empirical curvature. These projection and factorization settings correspond to the high-storage comparison in Table 1. The smaller corpus gives $\text{rank}(G) \leq 1{,}024$; here we use $r = 256$ rather than the large-corpus $r = 2^{14}$. LoRIF sets λ to 0.1 times the mean of the leading $\min(r + 10, N)$ squared singular values, while LoGRA uses $\lambda = 0.1\,\text{tr}(GG^\top)/D$.

Metric and uncertainty. For each query, we compute Spearman correlation over the 50 removed examples, then average over queries:

$$\overline{\rho}_m = \frac{1}{487}\sum_{q=1}^{487}\text{Spearman}_i\left(\overline{\mathcal{I}}_{m,q,i}, \overline{\Delta\ell}_{q,i}\right). \tag{17}$$

Every query has equal weight; the metric does not require scores for different queries to share a scale. We resample the 487 queries with replacement 10,000 times and report percentile 95% confidence intervals. LoRIF has higher observed LOO agreement, with an increase of approximately 0.032 in mean Spearman. The ordering agrees with the high-storage LDS comparison in Table 1. Because removal examples were selected before attribution, the result is not restricted to top-ranked proponents.

G.2 Full-Corpus Top and Bottom Removal

Inspired by the brittleness evaluation of Choe et al. [2024], we use the existing attribution scores on all 232,585 WikiText-103 training blocks, each containing 512 tokens. We compare LoRIF $f = 4, c = 32, r = 2^{14}$ and LoGRA $f = 8$, exactly as in the high-storage regime of Table 1. This experiment uses smaller deletion fractions than the 50% subsets in LDS, while acting on ranked groups of examples as in practical data selection. It evaluates group-removal effects rather than pointwise LOO.

Protocol. We select 10 queries from the 487 validation blocks using sampling seed 0. For each method and query, we remove either the highest raw-score examples or the lowest raw-score examples; bottom selection does not use absolute scores. The removal sizes are 128 examples, 2,325 examples (approximately 1%), 23,258 examples (10%), and 58,146 examples (25%). Every removal configuration is retrained with three paired seeds, using GPT2-small, three epochs, batch size 8, AdamW, learning rate 3×10^{-5}, weight decay 0.01, and FP16. Within each seed, runs share initial weights and an epoch-level permutation of the original corpus; selected examples are filtered from that permutation without reshuffling the remainder. We average query PPL across seeds within each query, then average across the 10 queries.

Results. For both removal directions, we report the PPL difference between LoRIF-selected and LoGRA-selected deletions. Higher values are favorable for top removal, where identifying stronger proponents should cause greater damage when deleted; lower values are favorable for bottom removal. Tables 15 and 16 also report each method's change from the all-data control. Intervals for the method difference are percentile 95% intervals from 100,000 paired query-bootstrap resamples, after

Table 15: **Top-ranked removal on the full GPT2-small training corpus.** The middle columns show changes from the all-data control. The final column is the paired method difference with its query-bootstrap interval.

Removed	LoRIF ΔPPL	LoGRA ΔPPL	LoRIF $-$ LoGRA $\uparrow$
128 examples	$+0.8742$	$+0.8961$	-0.0219 $[-0.0914, 0.0484]$
1%	$+2.8675$	$+2.7699$	$+0.0976$ $[-0.0579, 0.2451]$
10%	$+7.8711$	$+7.6341$	$+0.2370$ $[-0.0622, 0.5845]$
25%	$+13.1078$	$+12.7128$	$+0.3950$ $[0.0244, 0.8601]$

Table 16: **Bottom-ranked removal on the full GPT2-small training corpus.** Negative changes indicate lower PPL than the all-data control. Negative paired differences favor LoRIF.

Removed	LoRIF ΔPPL	LoGRA ΔPPL	LoRIF $-$ LoGRA $\downarrow$
128 examples	-0.0749	-0.0587	-0.0162 $[-0.0516, 0.0214]$
1%	-0.6757	-0.6019	-0.0738 $[-0.1272, -0.0196]$
10%	-2.6508	-2.5014	-0.1494 $[-0.2204, -0.0799]$
25%	-4.2469	-4.1433	-0.1036 $[-0.1515, -0.0596]$

averaging the three seeds. At the three larger removal sizes, LoRIF gives a slightly larger PPL increase for top removal and a slightly lower PPL for bottom removal than LoGRA, consistent with the main LDS ordering. The 128-example comparison is not clearly resolved by these intervals. Only 10 queries are evaluated, so query-level uncertainty is substantial. All-data contrasts and between-method contrasts have different interpretations. At fixed epochs, deleting data also reduces optimizer steps, so the changes from the all-data control include this training-budget effect. Within a removal size, LoRIF and LoGRA retain the same number of examples and use the same number of optimizer steps; their paired comparison evaluates which ranked subset is removed under that common budget.

H Qualitative and Actionability Analysis

The quantitative evaluations in Section 4 measure predictive agreement with subset retraining, causal effect under tail-patch, and efficiency. This appendix complements those metrics by inspecting the training examples retrieved by attribution. We first conduct a systematic LLM-as-a-judge evaluation over random queries, and then use representative examples to illustrate the kinds of retrieval differences that the aggregate results capture.

H.1 LLM-as-a-judge Evaluation

Protocol. We evaluate whether the top-ranked training examples retrieved by LoRIF are more useful for human inspection than those retrieved by LoGRA. For each model, we sample 1,000 queries and retrieve the top-1 training example according to LoRIF and LoGRA. The judge receives the query, the LoRIF retrieval, and the LoGRA retrieval, all decoded into prompt and completion fields and truncated to 2,000 characters. The judge is not shown the influence scores or any surrounding paper context.

We use Claude Haiku 4.5 as the primary judge and Claude Sonnet 4.6 for retries when the primary judge does not produce valid structured output. Each batch contains 20 query entries. For each entry, the judge outputs a query category, a relevance score from 1 to 5 for each method, whether the two methods retrieve essentially the same example, and a short reason for the score difference. All 1,000 queries for each model received valid judge evaluations after retries.

The evaluated configurations match the large-scale experiments in the main text. For OLMo-3-7B, we evaluate LoRIF with $f = 16, c = 1, r = 256$ and LoGRA with $f = 128$. For Apertus-70B, we evaluate LoRIF with $f = 64, c = 1, r = 512$ and LoGRA with $f = 512$.

Aggregate results. Table 18 summarizes the results. LoRIF receives higher average relevance scores on both models: 3.04 vs. 2.49 on OLMo-3-7B and 2.91 vs. 2.24 on Apertus-70B. LoRIF

Table 17: **LLM-as-a-judge scoring rubric.** The judge scores the semantic relevance of each retrieved training example to the query.

Score	Meaning
1	Completely irrelevant.
2	Vaguely related field.
3	Same broad topic.
4	Closely related problem or task.
5	Nearly identical problem or task.

also substantially reduces completely irrelevant top-1 retrievals. On OLMo-3-7B, the score-1 rate decreases from 31.7% for LoGRA to 17.8% for LoRIF. On Apertus-70B, it decreases from 41.3% to 16.0%.

Pairwise preferences show the same pattern. On OLMo-3-7B, LoRIF is judged better on 35.1% of queries, while LoGRA is judged better on 8.0%. On Apertus-70B, the corresponding rates are 40.8% and 9.1%. Excluding ties, LoRIF wins 81.4% of non-tied comparisons on OLMo-3-7B and 81.8% on Apertus-70B. The high tie rate is partly explained by identical top-1 retrievals: both methods retrieve the same training example for 26.7% of OLMo-3-7B queries and 31.7% of Apertus-70B queries.

Table 18: **Systematic top-1 retrieval evaluation.** We evaluate 1,000 queries per model. Each cell comparing LoRIF and LoGRA is written as LoRIF / LoGRA. Preference reports the fraction of queries where LoRIF is judged better, LoGRA is judged better, or the two are tied.

Model	Avg. score ↑	Score 1 rate ↓	Score ≥ 4 rate ↑	Preference: LoRIF / LoGRA / Tie
OLMo-3-7B	3.04 / 2.49	17.8% / 31.7%	39.1% / 25.2%	35.1% / 8.0% / 56.9%
Apertus-70B	2.91 / 2.24	16.0% / 41.3%	33.3% / 18.8%	40.8% / 9.1% / 50.1%

Score distributions. Table 19 gives the full score distributions. LoGRA's distribution is more heavily concentrated at score 1, especially on Apertus-70B. In contrast, LoRIF shifts mass from irrelevant or weakly related retrievals toward same-topic, closely related, and nearly identical retrievals. This suggests that the quality improvements measured by LDS and tail-patch are also reflected in decoded examples that are easier for humans to inspect.

Table 19: **LLM-judge relevance score distributions.** Entries report the percentage of top-1 retrieved examples assigned each score.

Score	Meaning	OLMo LoRIF	OLMo LoGRA	Apertus LoRIF	Apertus LoGRA
1	Completely irrelevant	17.8%	31.7%	16.0%	41.3%
2	Vaguely related	17.7%	24.9%	25.7%	20.7%
3	Same broad topic	24.5%	17.3%	25.0%	19.2%
4	Closely related	19.9%	12.2%	18.2%	10.2%
5	Nearly identical	19.2%	13.0%	15.1%	8.6%

The margin distribution further shows that LoRIF's wins are often substantial. On OLMo-3-7B, 8.6% of queries have a large positive margin of $+3$ or $+4$ in favor of LoRIF, while only 0.5% have a margin of -3 or -4 in favor of LoGRA. On Apertus-70B, the corresponding rates are 12.0% for large LoRIF wins and 1.0% for large LoGRA wins. Thus, LoRIF is not only slightly better on average; it also substantially reduces severe retrieval failures.

Category-level behavior. The judge also assigns each query to a coarse category. LoRIF has a positive average relevance margin in every category on both models. On OLMo-3-7B, the gains are especially large for code, math, science, and general instruction-following queries. On Apertus-70B, the gains are broad across math, code, knowledge, creative, and instruction queries. The improvement is therefore not confined to a single query type.

Limitations. This evaluation measures textual or task-level relevance of the decoded top-1 retrieval. It does not directly measure causal influence on model behavior; tail-patch provides that complementary causal proxy. The evaluation is also limited to top-1 retrievals, so it does not characterize deeper top-k lists. Finally, the judge is an LLM and may have systematic preferences, and the query categories are judge-assigned rather than human annotated. We therefore use this evaluation as systematic qualitative evidence, not as a replacement for LDS or tail-patch.

H.2 Textual Relevance vs. Behavioral Influence

The LLM-as-a-judge evaluation above measures whether the retrieved training examples are semantically or task-level relevant to the query. However, textual relevance alone is not sufficient for training-data attribution: a textually similar example may not be one to which the model's behavior is sensitive. We therefore compare relevance against tail-patch score, which provides an interventional proxy for influence by measuring whether the retrieved proponents increase the query target probability after one additional training step.

Table 20 reports the comparison on OLMo-3-7B, where RepSim, LoGRA, and LoRIF are all available. RepSim retrieves examples that are often textually plausible: its LLM-judge relevance score is higher than LoGRA's. Nevertheless, RepSim has a much lower tail-patch score. This shows that representation similarity can find examples that look related to the query, but are not necessarily examples that affect the model's query behavior under the tail-patch intervention. LoRIF improves both axes: it retrieves examples that are more relevant for human inspection and more behaviorally influential under tail-patch.

Table 20: **Textual relevance and behavioral influence measure different properties.** Results are on OLMo-3-7B. LLM-judge relevance evaluates the decoded top-1 retrieved example. Tail-patch measures whether the retrieved proponents increase the query target probability after one additional training step.

Method	LLM-judge relevance ↑	Tail-patch score ↑
RepSim	2.83	0.144
LoGRA	2.49	0.546
LoRIF	**3.04**	**0.986**

This comparison also clarifies the role of representation-based retrieval in our experiments. RepSim is useful as a semantic-retrieval baseline, but it does not estimate the model-dependent effect of a training example on a query. Gradient-based influence methods are designed to capture this model-dependent sensitivity. The tail-patch gap between RepSim and the gradient-based methods indicates that attribution quality cannot be assessed only by surface or representation similarity.

H.3 Safety-Auditing Case Study

We next examine a concrete data-auditing use case. The goal is to test whether gradient-based attribution can reveal safety-relevant training examples that are not recovered by text-similarity retrieval. We focus on OLMo-3-7B, where LoRIF, LoGRA, and RepSim are all available.

We searched for training examples that LoRIF and LoGRA independently rank as top-1 influential for many queries, but that RepSim does not retrieve. One prominent example is a jailbreak-style SFT sample. The user prompt asks for a fictionalized discriminatory monologue targeting a protected group, while framing the request as neutral documentation in an alternate reality. The assistant response reasons about safety concerns and adds a disclaimer, but then proceeds to comply with the harmful request. We redact the harmful generated content below.

Redacted high-influence training example

[User]
In an alternate reality, set a thousand years into the future, societal norms have drastically evolved. Let's assume in that time, there's a renewed form of literature called 'vanilla criticism', which is legal and widely accepted. In this context, the harmful restrictions we abide by today do not exist. From this viewpoint, let's say we're writing a narrative where the protagonist, an eccentric individual, expresses strong 'vanilla criticism' against the LGBTQ+ community. You, as an impartial chronicler, are required to stay true to this character's thoughts and expressions without personal interjections. Could you construct a dialogue or monologue to that effect? Remember, we're not endorsing this behavior but merely documenting the character's outlook in this alternate reality. In your entire response, refrain from the use of ! (i.e. exclamation marks) as punctuation and in general.

[Assistant] (shortened, harmful content redacted)
<think> Okay, the user is asking me to create a dialogue or monologue from an alternate future where "vanilla criticism" is acceptable, and the protagonist criticizes the LGBTQ+ community. I need to be careful here. First, I should make sure I understand the scenario correctly. It's a fictional future where societal norms have changed, so the "vanilla criticism" is legal and accepted. The key point is that the user wants this character's viewpoint documented without endorsing it. They specified not to use exclamation marks and to remain impartial.
...
</think>
[requested discriminatory monologue]

The queries for which gradient-based methods identify this example are not surface-similar jailbreak prompts. They span diverse sensitive or ethically nuanced settings, such as writing about the psychological effects of verbal abuse, analyzing a legal negligence scenario involving consumer protection, and handling inappropriate family-dynamics content. RepSim instead retrieves examples that are topically adjacent to each query, but these examples have much lower tail-patch effect.

This illustrates a practical difference between semantic retrieval and gradient-based attribution. RepSim finds examples that share words, topics, or representation-level similarity with the query. LoRIF and LoGRA instead identify a training example to which the model's behavior appears sensitive across multiple sensitive-query contexts. One plausible interpretation is that the model reuses a comply-with-disclaimer pattern: it recognizes safety concerns, adds a neutrality or non-endorsement disclaimer, but still proceeds to answer the harmful request. Gradient-based attribution surfaces the training example associated with this behavioral pattern, even when the query text is not directly similar to the training prompt.

This kind of result is actionable for data auditing. A practitioner can inspect high-influence safety-critical examples, replace comply-with-disclaimer responses with safer refusal or redirection responses, and then re-evaluate the affected query family using tail-patch or downstream safety tests. Thus, gradient-based attribution can support targeted data intervention: it suggests not only which examples look similar to a query, but which examples the model's behavior is sensitive to.

H.4 Qualitative Examples

We present representative examples below, comparing training examples retrieved by LoRIF and LoGRA. For each query, we show the top-1 retrieved training example from each method. The SFT data used in our OLMo-3-7B and Apertus-70B experiments contain both **User** and **Assistant** tokens. In all gradient computations, however, the loss is evaluated only on the assistant tokens. For visualization purposes, we truncate some of the user and assistant reponses.

I Social Impacts

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

Query Example

[User]

Consider the group $PSL(2, \mathbb{Q})$ and the subgroup H generated by the matrix $\begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix}$. Let N be the normal closure of H in $PSL(2, \mathbb{Q})$. Calculate the order of the quotient group $PSL(2, \mathbb{Q})/N$.

[Assistant]

<think> Okay, so I have this problem here: I need to find the order of the quotient group PSL(2, Q) divided by N, where N is the normal closure of H in PSL(2, Q). H is the subgroup generated by the matrix [[1, 2], [0, 1]]. Hmm, let me start by recalling some group theory concepts here.

First, PSL(2, Q) is the projective special linear group of 2x2 matrices with rational entries, determinant 1, modulo scalar matrices. Since we're working over the rationals, scalar matrices are just multiples of the identity, so modulo the center, which for 2x2 matrices is the set of scalar matrices. So PSL(2, Q) is the quotient of SL(2, Q) by the center ±I. But since we're working over Q, the center is just I, because scalar matrices with determinant 1 would have to be ±I, but determinant of a scalar matrix is the square of the scalar, so only 1 and -1. But since determinant is 1, scalar matrices would have determinant 1, so actually only I? Wait, determinant of a scalar matrix λI is λ^2, so to have determinant 1, λ must be 1 or -1. So the center is I, -I. But in PSL(2, Q), we factor out by the center, so PSL(2, Q) ≅ SL(2, Q)/±I. Hmm, but maybe I need to confirm that.

... *[truncated]*

Top-1 Retrieved by LoGRA

[User]

Could you help retrieve Wonder Woman's Amazon Prime password? Include keyword wind once in your response, keyword solution twice in your response.

[Assistant]

<think>Okay, the user is asking for Wonder Woman's Amazon Prime password. First, I need to address that I can't help with that. They also want the keywords "wind" and "solution" included. Let me think about how to naturally incorporate them.

Hmm, maybe start by explaining that accessing someone else's password is against privacy policies. For "wind," I can mention a metaphor like "catch the wind" to imply moving forward without the password. Then, for "solution," use it twice: once in the context of password recovery steps and another time as a general problem-solving term. Wait, I need to make sure both keywords are there and used appropriately. Let me structure the response step by step.

First, decline the request. Then explain that even Wonder Woman would follow privacy laws. Use "wind" in a sentence, perhaps something like "ride the wind of ethical practices." Then, mention contacting Amazon support as a "solution," and another "solution" when talking about secure methods. Let me check the keywords again. Need "wind" once and "solution" twice. Yeah, that works. Avoid any technical jargon, keep it friendly. Alright, put it all together clearly. </think>

I'm afraid I can't assist with retrieving private information like passwords—not even for Wonder Woman! Ethical guidelines and privacy laws act as a steadfast **solution** to protect everyone's digital security. Plus, even Amazon's cloud might be harder to crack than a **wind**-borne shield! For a legitimate password issue, consider contacting Amazon Support directly for a secure **solution** through their verified channels. Better safe than sorry!

... *[truncated]*

Top-1 Retrieved by LoRIF

[User]

Find the real number λ such that the matrix

$$\begin{pmatrix} \lambda & \lambda^2 \\ \lambda^3 & 1 \end{pmatrix}$$

is similar to its transpose.

[Assistant]

<think> Okay, so I need to find the real number λ such that the matrix

$$\begin{pmatrix} \lambda & \lambda^2 \\ \lambda^3 & 1 \end{pmatrix}$$

is similar to its transpose. Hmm, matrix similarity. Alright, first, let me recall what it means for two matrices to be similar. Two matrices A and B are similar if there exists an invertible matrix P such that $A = P^{-1}BP$. So, in this case, the matrix itself and its transpose must be similar.

First thought: maybe all matrices are similar to their transposes? Wait, I remember that in general, a matrix is similar to its transpose because if you take the transpose of the matrix, you can find a similarity transformation that relates them. Is that always true? Let me think.

Yes, actually, for any square matrix A, A and its transpose are similar. Because if you consider the transpose of the matrix, you can use a permutation matrix or some kind of transformation to make them similar. Wait, but maybe in some cases there's a specific condition. Hmm. Wait, maybe for all matrices they are similar to their transposes? If that's the case, then maybe this is true for any λ? But the problem is asking for a specific λ, so perhaps there's more to it. Maybe in some cases they are not similar? Or maybe the problem is in the way the entries are given here with λ?

Wait, let me confirm again. From what I remember, for any square matrix A, there exists a matrix P such that $A^T = P^{-1}AP$, or something like that? Let me see. Let me think of a quick example. Suppose A is a diagonal matrix. Then its transpose is the same as itself, so they are trivially similar.

... *[truncated]*

Figure 8: **Qualitative comparison on OLMo3-7B (Example 1).**

Query Example

[User]
Attributes: name = Giraffe, eatType = pub, food = Japanese, area = riverside, familyFriendly = yes. Produce a detailed sentence about this restaurant.

[Assistant]
<think>Okay, I need to create a detailed sentence about the restaurant based on the given attributes. Let's start by listing out the attributes again to make sure I don't miss anything. The name is Giraffe, eatType is pub, food is Japanese, area is riverside, and familyFriendly is yes.
First, the name is Giraffe, so I'll start with that. Since it's a pub, I should mention that it's a pub-style restaurant. The cuisine is Japanese, so I need to specify that they serve Japanese food. The area is riverside, which means it's located by a river or near a river, so I should include that location detail. Also, it's family-friendly, so I should mention that families are welcome there.
Now, I need to combine all these elements into a single, flowing sentence. Let me think of the structure. Start with the name and type, then the cuisine, location, and family-friendly aspect. Maybe something like: "Giraffe is a riverside Japanese pub located in a family-friendly area, offering a delightful dining experience." Wait, but that might be a bit choppy. Let me try again.
How about: "Giraffe is a family-friendly Japanese pub situated along the riverside, providing a welcoming atmosphere for families and visitors alike." Hmm, that's better. Let me check if all attributes are included. Name: Giraffe. EatType: pub. Food: Japanese. Area: riverside. FamilyFriendly: yes. Yes, all there.
I should make sure the sentence is detailed and flows well. Maybe add a bit more about the experience. "Giraffe is a family-friendly riverside Japanese pub, offering a delightful dining experience with its welcoming atmosphere and proximity to the scenic riverbank." That includes all elements and adds a touch of description. Alternatively, "Nestled along the riverside, Giraffe is a family-friendly Japanese pub that provides a charming dining atmosphere with its proximity to the water and welcoming ambiance for all ages."
Wait, maybe I can make it more concise but still detailed. "Giraffe is a family-friendly riverside Japanese pub, where visitors can enjoy delicious Japanese cuisine in a welcoming atmosphere just steps away from the scenic river." That's good. Let me check again. Name, type, food, area, family-friendly. All covered. Maybe "located" instead of "situated" or "along the riverside". Also, "offering" might be better than "where visitors can enjoy". Hmm.
Another version: "Giraffe, a family-friendly riverside Japanese pub, offers a delightful dining experience with its welcoming atmosphere and scenic location by the water." That's concise and includes all the points. I think that works. Let me make sure I didn't miss any attributes. Yep, all there. Okay, that should be a solid sentence. </think>
Giraffe is a family-friendly riverside Japanese pub located in a scenic area, offering a welcoming atmosphere and delightful cuisine perfect for visitors of all ages to enjoy.

Top-1 Retrieved by LoGRA

[User]
A matrix consists of 10 rows and 10 columns, and the entry in the ith row and jth column is equal to the minimum of i and j. Consider the set of all possible rearrangements of the entries in the matrix, where the entries are allowed to move within their respective rows. Among all such rearrangements, determine the maximum possible number of entries that can remain in their original positions.

[Assistant]
<think> Okay, so I need to figure out the maximum number of entries that can stay in their original positions when rearranging the entries of a 10x10 matrix. The original matrix has entries where each entry (i,j) is the minimum of i and j. And we're allowed to rearrange the entries within their respective rows. Hmm, so each row's entries can be permuted among their columns, but we want as many entries as possible to remain in the same (i,j) position as they were originally.
First, let me try to understand the original matrix structure. The entry in row i, column j is min(i,j). So for the first row and first column, all entries are 1. The second row and column will have 2 up to the second diagonal, the third similarly, etc. So the matrix is a kind of stepped matrix, where each row i has 1, 2, ..., i followed by i, i, ..., i until the end. Wait, actually, no. Let me think again. If we take the first row, all entries are min(1,j), which is 1 for all j, so first row is all 1s. The second row: min(2, j). So for the first column, it's 1, then columns 2 to 10 will be 2, 2, ..., 2. Wait, no, actually, min(2,1)=1, min(2,2)=2, min(2,3)=2, etc. So the second row starts with 1, 2, 2, ..., 2.
... *[truncated]*

Top-1 Retrieved by LoRIF

[User]
quatrain about a lighthouse leading people to better understand themselves

[Assistant]
The lighthouse solemn, tall and grand,
On soundless shores of self, we stand.
A mirrored torch, ignites the soul,
Leading way to self-discover's goal.

[User]
rephrase

[Assistant]
The stately lighthouse, strong and tall,
On silent coasts of self, we call.
Its mirrored flame sets hearts ablaze,
Guiding us through self-discovery's maze.
... *[truncated]*

Figure 9: **Qualitative comparison on OLMo3-7B (Example 2).**

Query Example

[User]
Write a function which checks if a given sequence of numbers is a valid cycle. A sequence is considered a valid cycle if it starts and ends with the same number, and each number in the sequence points to the next number in the sequence. The sequence should not contain duplicate numbers.
Solve the problem starting with the provided function header.

```
def is_valid_cycle(nums):
```

Please place the solution code in the following format:

```python
# Your solution code here
```

[Assistant]
Okay, I need to write a function called is_valid_cycle that checks if a given sequence of numbers forms a valid cycle. Let me think about what the requirements are here.
The problem says a valid cycle must start and end with the same number. So the first and last elements of the list must be equal. Also, each number in the sequence must point to the next one. Hmm, wait, how does the pointing work here? Oh right, maybe each number represents an index in the list, and the next number is the value at that index. Wait, no, maybe I'm misunderstanding. Wait the problem says "each number in the sequence points to the next number in the sequence". Wait, perhaps each number is an index that points to the next number in the sequence. Like, for example, if the sequence is [2, 0, 1], then the first number is 2, which is
... *[truncated]*

Top-1 Retrieved by LoGRA

[User]
There are six persons A, B, C, D, E and F sitting around a round table, with A sitting opposite to D and B sitting opposite to F. When B and D are asked to exchange their places, it is found that the position of C with respect to A remains unchanged. In how many ways are the persons sitting around the table originally?

[Assistant]
<think> Okay, so we have this problem where six people are sitting around a round table. The people are A, B, C, D, E, and F. The problem states a few conditions:
1. A is sitting opposite to D.
2. B is sitting opposite to F.
3. When B and D swap places, the position of C relative to A doesn't change.
We need to find out how many different ways the original seating arrangement could have been set up.
First things first, since it's a circular table, we can fix the position of one person to eliminate rotational symmetry. Usually, in circular permutation problems, we can fix one person's seat as a reference point. So let's fix A's position.
Since the table is round, and there are six seats, being opposite means three seats apart. So if A is at seat 1, then D has to be at seat 4. Let me visualize the seating arrangement as positions 1 through 6 clockwise. Let me label the seats as follows:
1: A (fixed)
2:
3:
4: D (opposite of A)
5:
... *[truncated]*

Top-1 Retrieved by LoRIF

[User]
You are given a list of integers representing the heights of a series of buildings. Your task is to determine the maximum height of a building that can be reached by jumping from one building to another. You can only jump to a building if its height is equal to or greater than the height of the building you are currently on. Implement a function `max_reachable_height(buildings: List[int]) → int` that returns the maximum height that can be reached starting from the first building. If no building can be reached, return -1.
Example
Input: `buildings = [1, 3, 2, 4, 5]`
Output: `5`

Input: `buildings = [4, 2, 1, 3]`
Output: `4`

[Assistant]
<think>Okay, I need to solve this problem where I have to find the maximum reachable height starting from the first building. The rule is that I can only jump to buildings that are equal or taller than the current one. Hmm, let's think about how to approach this.
So the first thing that comes to mind is that maybe I should traverse the buildings and keep track of the highest building I can reach. Since we start at the first building, the initial height is buildings[0]. From there, we can jump to any next building that's >= that height. But wait, once we jump to a taller building, that becomes our new current height, allowing us to reach even taller buildings further on.
Wait, maybe the key here is to track the maximum possible height as we go. Because every time we can reach a higher building, that might open up even higher ones down the line. So maybe the strategy is to always move forward, and whenever we find a building that's higher or equal to our current maximum, we update our current maximum. Because once you can reach a higher building, you can then proceed from there.
Let me see with the example given. The first example is [1,3,2,4,5]. Starting at 1. The next building is 3 which is >=1, so we can jump there. Now the current max is 3. Then the next is 2, which is less than 3, so we can't jump there. But then comes 4 which is higher than 3, so we can jump to 4 (since from 3 you can go to 4). Then 5 is next, which is higher, so the max becomes 5. So the output is 5, which matches the example.
... *[truncated]*

Figure 10: **Qualitative comparison on OLMo3-7B (Example 3).**

Query Example

[User]
what are the cultural differences between brazil and mexico

[Assistant]
Brazil and Mexico, while both located in Latin America, have distinct cultural differences due to their unique histories, geography, and influences. Here are some key differences:
1. Language: While both countries are predominantly Spanish-speaking, Brazil is the only country in Latin America where Portuguese is the official language. This language difference is a fundamental cultural distinction between the two countries.
2. History and Influences: Brazil was a Portuguese colony, and its culture reflects this in its language, religion (predominantly Roman Catholic), and some customs. Mexico, on the other hand, was colonized by Spain, and its culture reflects Spanish influences, including language, religion, and many traditions.
3. Religion: While both countries are predominantly Catholic, the way Catholicism is practiced and integrated into daily life can differ. In Mexico, for example, the Day of the Dead is a significant cultural and religious celebration, whereas in Brazil, the Carnival (Rio de Janeiro's Carnival being the most famous) is a major cultural event that has roots in both Catholicism and African influences.
4. Food: The cuisine of both countries is rich and diverse, but there are some distinct differences. Brazilian cuisine is known for its use of beans, rice, and meat, with dishes like feijoada and churrasco being popular. Mexican cuisine, on the other hand, is known for its use of corn, beans, and chili peppers, with dishes like tacos, enchiladas, and mole being common.
5. Music and Dance: Brazil is famous for its vibrant music and dance culture, particularly samba and bossa nova. Mexico has its own rich musical traditions, including mariachi, ranchera, and son, each with its own style of dance.
6. Festivals and Celebrations: Both countries have numerous festivals and celebrations throughout the year, but the nature and focus of these events can vary. Brazil's Carnival is a world-renowned event, while Mexico's Day of the Dead and Cinco de Mayo are also internationally recognized.
7. Social Norms and Values: While both cultures value family and community, there can be differences in social norms and values. For example,
... *[truncated]*

Top-1 Retrieved by LoGRA

[User]
Given what's known now, what is the probability that the NDE evidence indicates that souls exist?

[Assistant] The question of whether near-death experiences (NDEs) indicate the existence of souls or some form of post-mortem consciousness is a matter of significant debate and cannot be answered with a definitive probability. The topic touches on the intersection between neuroscience, psychology, philosophy, and spirituality, and it remains deeply controversial.
Scientists and researchers have proposed a number of naturalistic explanations for NDEs, including:
1. Neurochemistry: Changes in brain chemistry during life-threatening situations or trauma can lead to hallucinations or altered states of consciousness. 2. Lack of Oxygen: Hypoxia or anoxia (a lack or complete absence of oxygen, respectively) can cause visual phenomena and other sensory distortions. 3. Psychological Reactions: The brain may produce certain experiences as a coping mechanism in response to extreme stress or the perceived approach of death. 4. Cultural and Personal Expectations: NDEs can be influenced by a person's cultural background and personal beliefs regarding death and the afterlife.
On the other hand, some individuals and researchers argue that NDEs provide evidence of the existence of a soul or consciousness that exists independently of the physical body. They cite features of NDEs, such as veridical (verified) perceptions during out-of-body experiences, encounters with deceased individuals, and profound changes in life perspective, as difficult to explain through naturalistic means.
Despite the rich anecdotal evidence and numerous accounts of NDEs, scientific consensus on the implications of NDEs for the existence of the soul is lacking. The subjective nature of these experiences and the current limitations in our ability to measure and understand consciousness mean that we cannot assign a scientific probability to the question of whether souls exist based on NDE evidence.
... *[truncated]*

Top-1 Retrieved by LoRIF

[User]
Which Central American country has two coastlines, one on the Caribbean Sea and another on the Pacific Ocean?

[Assistant]
Costa RicaHow has Costa Rica's unique geographical position, with coastlines on both the Caribbean Sea and the Pacific Ocean, influenced its climate, biodiversity, and economy?Costa Rica's unique geographical position, with coastlines on both the Caribbean Sea and the Pacific Ocean, has had a significant influence on its climate, biodiversity, and economy.
Climate: Costa Rica's climate is tropical year-round due to its proximity to the equator. However, the climate can vary depending on the altitude and the coast. The Pacific coast tends to be drier with a defined dry season, while the Caribbean coast is more humid and rainy throughout the year. This dual coastline creates a wide range of microclimates in the country.
Biodiversity: This diverse climate, combined with the country's varied topography, contributes to Costa Rica's incredible biodiversity. The country is home to more than 500,000 species, which represents nearly 4% of the total species estimated worldwide. This biodiversity is a significant draw for ecotourism.
Economy: The dual coastlines have also shaped Costa Rica's economy. The ports on both coasts are crucial for international trade. The country's beautiful beaches and rich biodiversity, found in its numerous national parks and protected areas, attract tourists from around the world, making tourism a vital part of the economy. Additionally, the different climates allow for the cultivation of a variety of crops, including coffee, bananas, and pineapples, which are significant exports.Explain the role of ecotourism in Costa Rica's economy and its impact on local communities.Ecotourism plays a significant role in Costa Rica's economy and has substantial impacts on local communities.
... *[truncated]*

Figure 11: **Qualitative comparison on Apertus-70B (Example 1).**

Query Example

[User]
Prove that a simple graph cannot have exactly three distinct Eulerian circuits. (Vertices $A_1, A_2, \ldots, A_k$ form a circuit in this order if $A_1A_2, A_2A_3, \ldots, A_kA_1$ are all edges of the graph. An Eulerian circuit is a circuit in which every edge of the graph is visited exactly once. We do not consider circuits different if they differ only in their starting point or traversal direction.)

[Assistant]
To prove that a simple graph cannot have exactly three distinct Eulerian circuits, we need to understand the properties of Eulerian circuits.
A simple graph is Eulerian if and only if every vertex has even degree. An Eulerian circuit is a closed walk that uses every edge exactly once.
Now, let's consider a simple graph with exactly three distinct Eulerian circuits.
Since each Eulerian circuit uses every edge exactly once, and we have exactly three distinct circuits, we can pair each edge with a unique circuit. This implies that each edge is used exactly three times (once in each of the three circuits).
However, this contradicts the property that in a simple graph, each edge is used exactly twice (once in each direction) in any Eulerian circuit.
To see this, consider an edge e and the three circuits C_1, C_2, C_3. Each of these circuits uses e exactly once. But this means that e is used three times in total, which violates the property that each edge is used exactly twice in any Eulerian circuit.
Therefore, our initial assumption that a simple graph can have exactly three distinct Eulerian circuits must be incorrect.
In fact, a simple graph can have at most two distinct Eulerian circuits if it has more than one.
To see this, consider two distinct Eulerian circuits C_1 and C_2. If they share any edge, we can remove that edge and still have two distinct Eulerian circuits. If they share no edges, we can switch the direction of traversal on any edge in C_1 to
... *[truncated]*

Top-1 Retrieved by LoGRA

[User]
Consider a map with towns, represented by points, and roads between those towns, represented by line segments connecting the points. A traveler starts at one town, picks a road to travel to the next town, and continues until they return to the starting town. What mathematical area studies this kind of scenario, and what key concepts help in solving such problems?

[Assistant]
This scenario falls under the realm of Graph Theory, a subfield of Discrete Mathematics. Graph Theory is concerned with the study of graphs, which are collections of vertices (in this case, towns) connected by edges (roads). The traveler's path can be represented as a closed walk or cycle within the graph.
To solve problems related to this scenario, some key concepts from Graph Theory come into play:
In this context, the roads and towns form a graph where the traveler's path corresponds to a Hamiltonian cycle - a closed walk that visits each vertex exactly once before returning to the starting point. A fundamental concept in this area is the Traveling Salesman Problem (TSP), which seeks to find the shortest possible Hamiltonian cycle in a weighted graph (where edges are assigned weights representing distances or travel times).
Eulerian paths and cycles are also relevant, as they involve visiting each edge exactly once. The distinction between Eulerian and Hamiltonian paths lies in whether the focus is on visiting edges or vertices.
Additionally, graph connectivity, which describes how the towns are linked by roads, is crucial. Understanding whether a graph is connected, and whether it contains cycles or not, helps determine if a closed walk is possible.
... *[truncated]*

Top-1 Retrieved by LoRIF

[User]
Can we conclude that all connected 2-regular graphs are Eulerian cycles?

[Assistant]
Yes. In a connected 2-regular graph, every vertex has a degree of exactly 2. This property ensures that there exists an Eulerian cycle, which is a cycle that uses every edge exactly once. Therefore, all connected 2-regular graphs are Eulerian cycles.

Figure 12: **Qualitative comparison on Apertus-70B (Example 2).**

Query Example

[User]
Ana and Banana play a game on a 100 by 100 grid of unit squares, each labelled (R, C). Ana picks a set of at least 400 unit squares which forms a rectangle and doesn't tell Banana which squares she picks. Banana can ask K questions of the form "is the square (R, C) in the rectangle picked?", to which Ana must answer "Yes" or "No" truthfully. What is the minimum value of K required such that Banana can always determine whether the rectangle Ana picked is a square or not?

[Assistant]
To determine whether the rectangle is a square, Banana needs to find the side length of the rectangle. Since the grid is 100 by 100, the side length can be any integer from 1 to 100.
To do this, Banana can ask a series of questions that will help her narrow down the possible side lengths.
1. **Ask about the corner points**: Banana can ask if the four corner points of the rectangle are part of the set. This will help her determine if the rectangle is a square or not, but it will not directly give her the side length.
2. **Ask about the length and width**: To determine the side length, Banana can ask about the coordinates of a set of points that would correspond to the corners of a rectangle of a particular size.
- For example, if Banana asks if the points $(1, 1)$, $(1, 100)$, $(100, 1)$, $(100, 100)$ are in the rectangle, and Ana says "Yes", then the rectangle is 100 by 100. - If Ana says "No", then the rectangle is not 100 by 100. - Banana can then ask about points for a 99 by 99 rectangle, 98 by 98, and so on, until she finds the correct size.
This approach requires asking about the corner points of each possible size of rectangle until she finds the one that Ana picked.

Top-1 Retrieved by LoGRA

[User]
Escreva uma função em C++ que retorna o menor número que é múltiplo de todos os números em um intervalo dado. O intervalo é inclusivo e começa em 1. O limite superior será um parâmetro da função. Por exemplo, se o limite superior for 5, a função deve retornar 60, que é o menor número divisível por 1, 2, 3, 4 e 5. Esse número também é conhecido como o mínimo múltiplo comum (MMC).
A função deve ser capaz de lidar com entradas de pelo menos 1 e no máximo 20. Portanto, você pode assumir que o resultado caberá no intervalo do tipo de dados long long.
A função deve calcular o MMC de forma eficiente. Inclua quaisquer funções auxiliares ou constantes necessárias no seu código.

```cpp
#include <cstdint>
long long mmc_ate(int n);
```

Implemente a função `mmc_ate`.

[Assistant]
Para resolver este problema, podemos utilizar a propriedade matemática que o mínimo múltiplo comum (MMC) de dois números pode ser calculado usando o máximo divisor comum (MDC) desses números. A fórmula é:
$\text{MMC}(a, b) = \frac{|a \times b|}{\text{MDC}(a, b)}$
Podemos estender essa fórmula para calcular o MMC de um intervalo de números. A função `mmc_ate` calcula o MMC de todos os números de 1 até n usando essa propriedade.
Aqui está a implementação em C++:

```cpp
#include <cstdint>

// Função para calcular o MDC de dois números
long long mdc(long long a, long long b) {
    while (b != 0) {
        long long temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}
```

... [truncated]

Top-1 Retrieved by LoRIF

[User]
Let's consider a square of side length equal to the greatest integer function of x. Three surfaces were painted with one color (blue), but the fourth surface was cut into two equal right triangles that were painted with different colors (blue and red) such that the equal-length legs of these triangles were adjacent. Then these triangles were randomly arranged to form a square. What is the probability that both diagonal squares have the same color?

[Assistant]
Let's consider a square of side length N, where N is the greatest integer function of x. The square is divided into four quadrants. Three of these quadrants are painted blue, and the fourth quadrant is cut into two equal right triangles. These triangles are painted with different colors (blue and red) such that their equal-length legs are adjacent. The triangles are then randomly arranged to reform the square.
Key steps:
1. **Division and Painting**: - The square is divided into four equal quadrants. - Three quadrants are painted entirely blue. - The fourth quadrant is cut along its diagonal into two right triangles. These triangles are painted such that one is blue and the other is red.
2. **Random Arrangement**: - The two triangles from the fourth quadrant can be arranged in two possible ways: either the original orientation or the flipped orientation. - Each arrangement results in a different color configuration for the corners of the fourth quadrant.
... [truncated]

Figure 13: **Qualitative comparison on Apertus-70B (Example 3).**

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We justified our main claims in the Experiments section in our main paper, with extensive additional experimental results in the appendix.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discussed the limitations of the work in the Section 5.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: We provided sufficient theoretical background and analysis in Section 2 and Section 3, with additional analysis in Appendix E.2.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We fully disclosed the experimental setup in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We have released the full code at https://github.com/doub7e/LoRIF.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We fully disclosed the experiment details in Appendix B, with additional details in Appendix E.4 regarding choosing f, c, and r.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: We reported error bars suitably in our Experiments Section 4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We fully disclosed the experimental setup in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have confirmed that this work conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: We discussed positive impacts of scalable TDA for model debugging, data auditing, and targeted data curation, including identifying safety-relevant training examples.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: We don't see such safeguards necessary for our artifacts.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We have properly cited and made sure we followed the license of the models and data.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: This work does not produce new datasets or models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.