Training Data Attribution (TDA) with Influence Functions

Background: Influence Functions

- *Influence functions* are a classical idea from robust statistics (Hampel, 1974), which was introduced to deep learning by Koh and Liang (2017).

- Assume we have a training dataset $\{z_i\}_i^N$. E.g., for supervised learning, $z_i = (x_i, y_i)$. We fit the parameters using empirical risk minimization:

$$\boldsymbol{\theta}^\star \in \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}, \mathcal{D}) = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta})$$

- We want to understand the effect of *removing (or adding) a training example* z. We parameterize the training set by z's weight ϵ, and see how the optimal solution varies (i.e., the *response function*):

$$r(\epsilon) := \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(z_i, \boldsymbol{\theta}) + \epsilon \mathcal{L}(z, \boldsymbol{\theta})$$

Background: Influence Functions

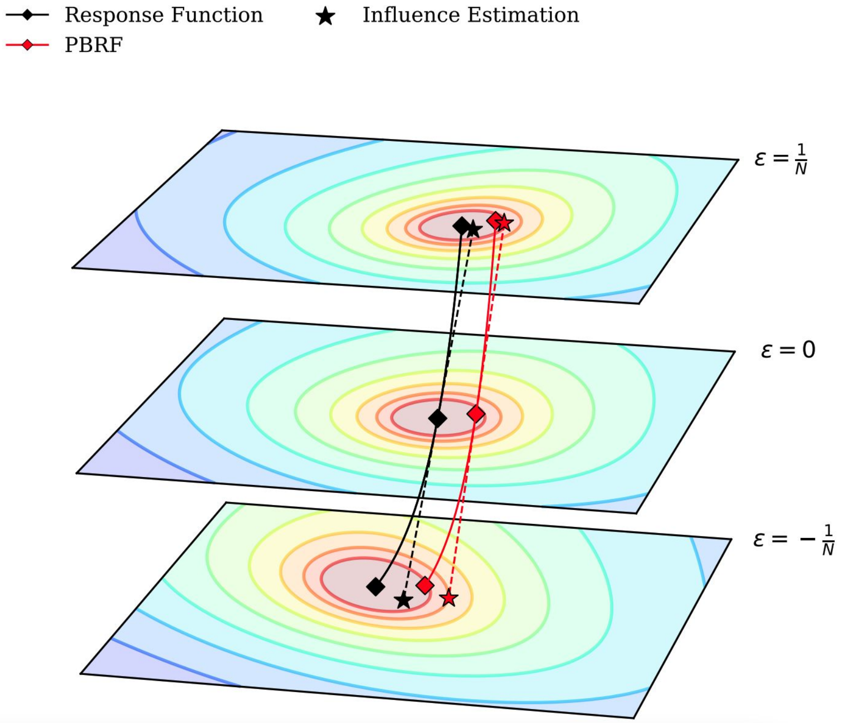
- The influence of z on $\boldsymbol{\theta}^\star$ is defined as the *first-order Taylor approximation* of the response function.

- Under certain regularity conditions, this can be computed using the *Implicit Function Theorem*:

$$\mathcal{I}_{\boldsymbol{\theta}^\star}(z) := \frac{\mathrm{d}r}{\mathrm{d}\epsilon} = -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z, \boldsymbol{\theta}^\star)$$

- Thus, we can approximate the change in parameters as:

$$r(\epsilon) - \boldsymbol{\theta}^\star \approx -\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}}\mathcal{L}(z, \boldsymbol{\theta}^\star)\epsilon$$



Background: Influence Functions

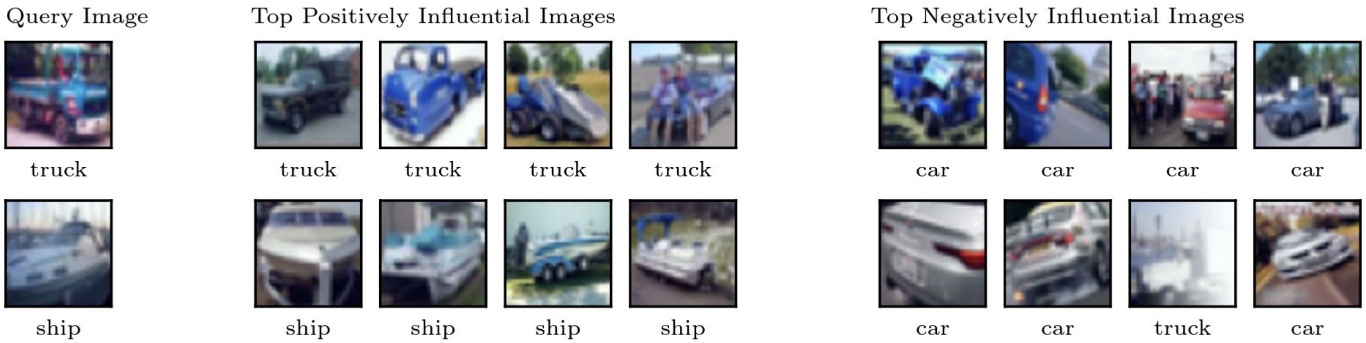
- Using the Chain Rule for Derivatives, we can approximate *the change in the measurable quantity* of query data point z_q by perturbing a weight of a training data point z:

$$f(z_q, r(\epsilon)) - f(z_q, \boldsymbol{\theta}^\star) \approx -\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z, \boldsymbol{\theta}^\star)\epsilon$$

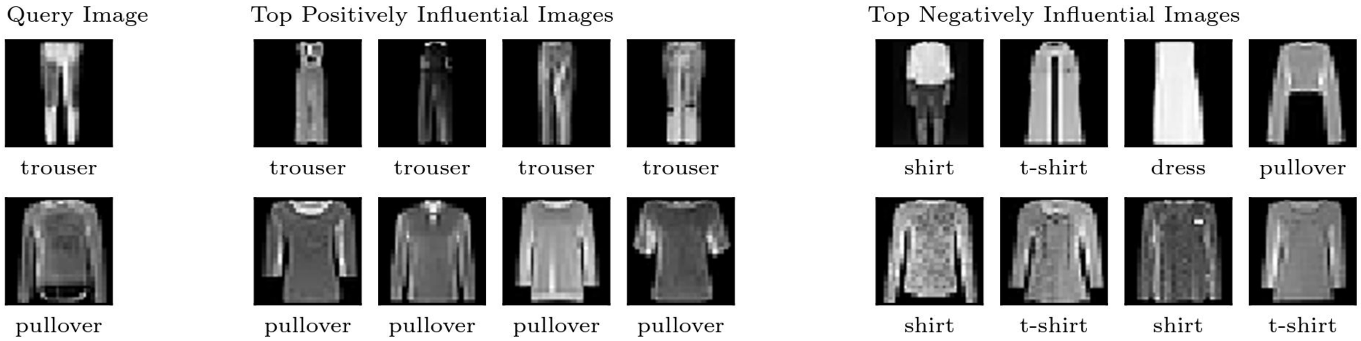
- The *measurement* f is chosen based on metrics relevant to the analysis, such as *loss*, *margin*, or *log-probability*.
  - When the measurement is defined as the loss, a higher absolute score signifies a more substantial change in the query loss when the data point z is excluded from (or added to) the training dataset.

Examples of Highly Influential Data Points

CIFAR-10



FashionMNIST



Influence Functions: Scalability Challenges

- Given a query data point z_q, we aim to identify influential training data points. The influence function computation process can be broken down into *two* stages:

$$\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)$$

Influence Functions: Scalability Challenges

- Given a query data point z_q, we aim to identify influential training data points. The influence function computation process can be broken down into *two* stages:

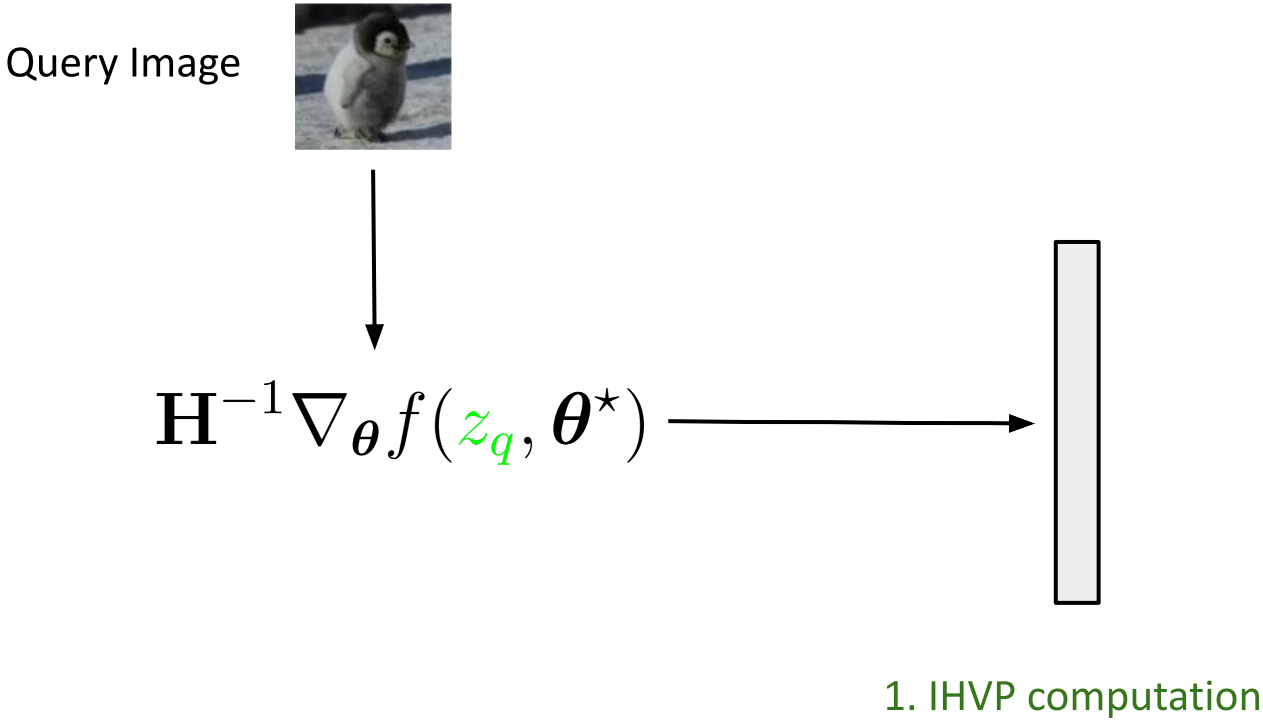
$$\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)$$

Influence Functions: Scalability Challenges

- Given a query data point z_q, we aim to identify influential training data points. The influence function computation process can be broken down into *two* stages:

$$\underbrace{\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}_{\substack{\text{2. Dot product computations with} \\ \text{all candidate training data points}}}$$

Influence Functions: Scalability Challenges

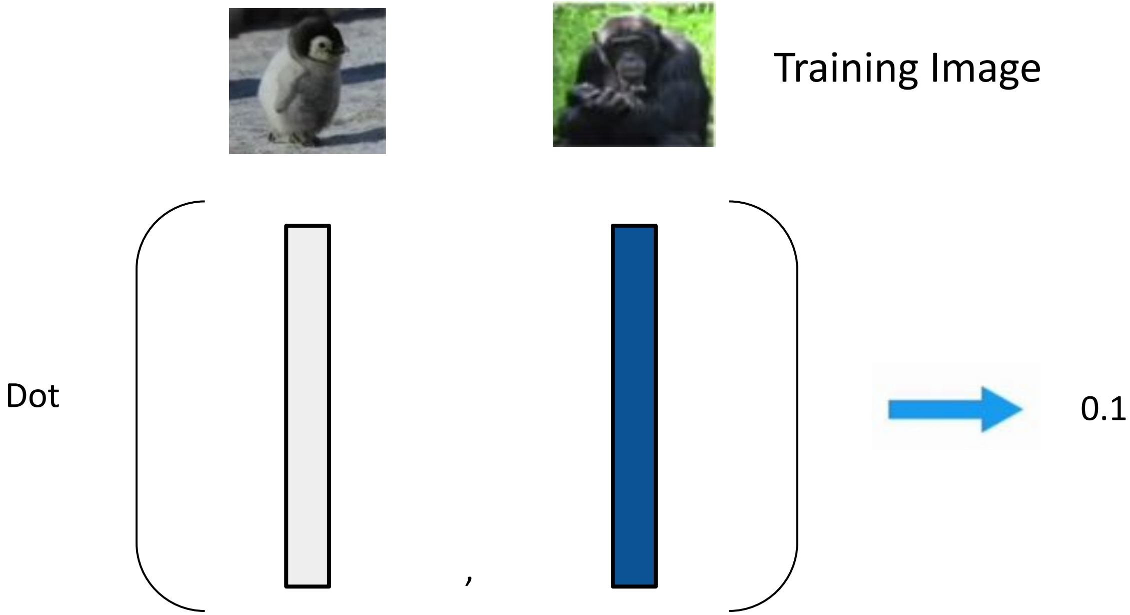


Query Image

$$\mathbf{H}^{-1}\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)$$

1. IHVP computation

Influence Functions: Scalability Challenges



Training Image

Dot

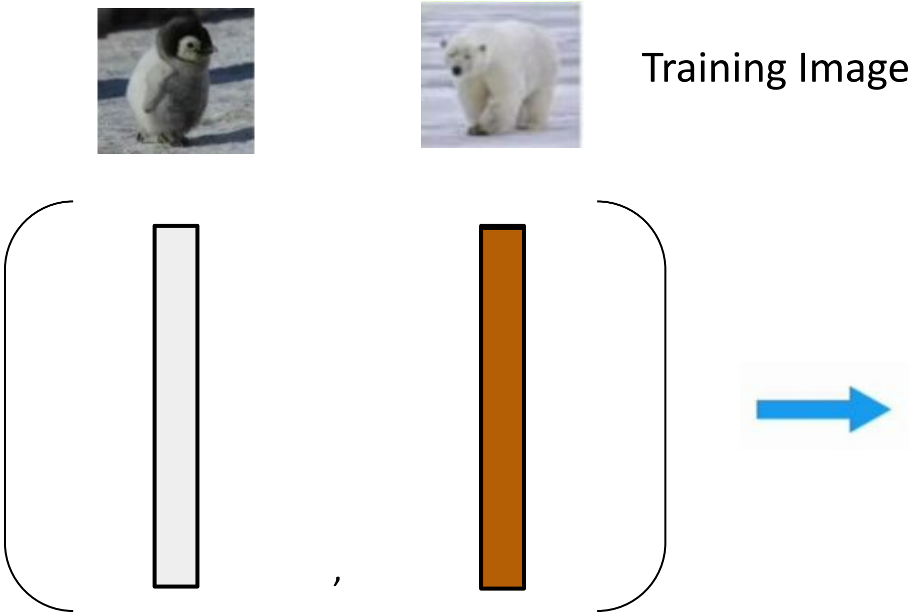
0.1

IHVP

Train Gradient 1

2. Dot product computations with all candidate training data points

Influence Functions: Scalability Challenges



Training Image

Dot

-0.4

IHVP

Train Gradient 2

2. Dot product computations with all candidate training data points

Influence Functions: Scalability Challenges

$$\underbrace{\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}_{\text{2. Dot product computations with all candidate training data points}}$$

- **Challenge 1: IHVP computation**
  - The Hessian **H** has dimensions equal to the square of the model's parameter count, making explicit computation infeasible for large models.
  - In our past works, we have employed *efficient IHVP approximation techniques*, such as LiSSA and EKFAC.
    - Using EKFAC, we successfully scaled influence functions with to LLMs with over 52 billion parameters.

Influence Functions: Scalability Challenges

$$\underbrace{\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}_{\text{2. Dot product computations with all candidate training data points}}$$

- **Challenge 2: Dot product computations**
  - This stage requires computing the per-sample gradient for *all candidate training data points*.
  - For LLM pretraining, this process is at least as expensive as the cost of pretraining the model itself (1 epoch training).
  - Furthermore, this needs to be *repeated* for each query data point z_q.

Influence Functions: Conceptual Challenges

- The classical formulation of influence functions just described does not quite apply to modern neural networks.
    - Assumes **H** is invertible, while neural network training is often underspecified.
    - Assumes that we have found the *optimal solution* $\boldsymbol{\theta}^\star$ and TDA is performed on this optimal solution.
    - In practice, the gradients and Hessian computed using *the final parameters* from a single training run (rather than the optimal solution).

- Moreover, the classic formulation of influence functions cannot incorporate the *details of the training process*:
    - Location of a data point of interest z appeared during training.
    - Implicit bias of optimizers (e.g., the use of SGD vs. Adam).
    - Learning rate schedules.

What is Your Data Worth to GPT? LLM-Scale Data Valuation with Influence Functions

Sang Keun Choe, Hwijeen Ahn, **Juhan Bae** , Kewen Zhao, Minsoo Kang,
Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell,
Teruko Mitamura, Jeff Schneider, Eduard Hovy, Roger Grosse, Eric Xing

Motivation: Gradient Caching

- **Recall:** Training gradients need to be *recomputed* for each new query.

$$\underbrace{\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}_{\substack{\text{2. Dot product computations with} \\ \text{all candidate training data points}}}$$

Motivation: Gradient Caching

- **Recall:** Training gradients need to be *recomputed* for each new query.

$$\underbrace{\underbrace{\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1}}_{\text{1. IHVP computation}} \boxed{\nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}}_{\text{2. Dot product computations with all candidate training data points}}$$

+ Save the training gradient to reuse it for new query

- **Idea:** Can we *save all individual training gradients* in storage to avoid the recomputation of training gradients for new query?
    - Once the training gradients are all saved on storage, we can use *approximate nearest neighbor search* to compute influence scores for all training data points.

Motivation: Gradient Caching

- **Problem:** High memory costs due to an (extremely) *high-dimensional nature of gradients*

$$\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star)^\top \mathbf{H}^{-1} \underbrace{\nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star)}_{\text{dim(grad) = \# params}}$$

- For 8B model (e.g., Llama3-8B) with 100,000 training data points, we need 3.2 PB (or 3,200 TB) of storage to save all training gradients.
  - Loading a 32 gigabyte (GB) vector for each training data point incurs a significant disk I/O overhead.

Gradient Projection

- **Idea:** One strategy to address the scalability challenge is to project the gradients onto a low-dimensional space and compute influence scores within the subspace spanned by the projection matrix.

$$(\mathbf{P}\nabla_{\boldsymbol{\theta}} f(z_q, \boldsymbol{\theta}^\star))^\top (\mathbf{P}\mathbf{H}\mathbf{P}^\top)^{-1} (\underbrace{\mathbf{P}}_{\substack{\text{Projection matrix of size } k \times m \\ \text{(where } k << m\text{)}}}\nabla_{\boldsymbol{\theta}} \mathcal{L}(z_i, \boldsymbol{\theta}^\star))$$

- **Advantages:**
  - Low-rank projection enables *writing projected gradients for all training data to disk* once and simply reading them as new query data point arrives (without costly recomputations).
    - Llama3-8B & 100,000 training data: 3.2PB ⟶ 4GB (when k = 10,000)
  - Reduced the Hessian inversion cost.
  - (Future) Reduced influence analysis costs.

Gradient Projection

- **Problem:** The above benefits come at the cost of an additional gradient projection
  - Time complexity for computing the projected per-sample gradient: $O(bkm)$
    - *b*: Batch size
    - *k*: Projection dimension
    - *m*: Model dimension
  - Space complexity for the projection matrix: $O(km)$
    - E.g., the matrix size for an 8B model and the 4k projection dimension: 128TB!

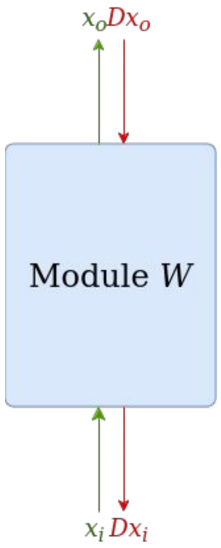
Gradient Projection

- **Problem:** The above benefits come at the cost of an additional gradient projection
  - Time complexity for computing the projected per-sample gradient: $O(bkm)$
    - *b*: Batch size
    - *k*: Projection dimension
    - *m*: Model dimension
  - Space complexity for the projection matrix: $O(km)$
    - E.g., the matrix size for an 8B model and the 4k projection dimension: 128TB!

- **Summary so far**:
  - Data influence analysis can be seen as vector similarity analysis in gradient space.
  - Gradient is too high-dimensional → projection is necessary
  - A projection cost is too high → ?

Efficient Gradient Projection (LoGra)

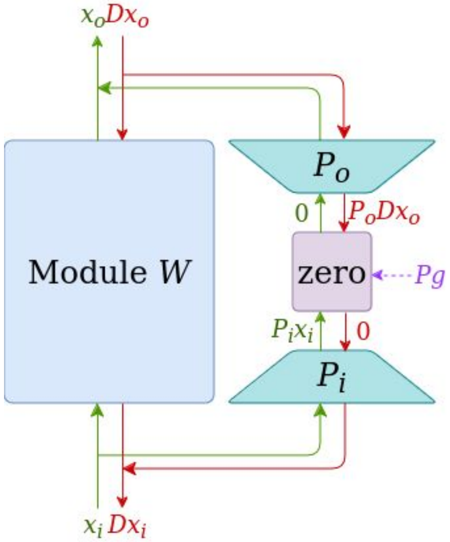
- **Observation:** Gradient *DW* in backpropagation is Kronecker-product (or 2D) structured:



$$\mathrm{vec}(\mathcal{D}W) = \sum_{t=1}^{T} x_{i,t} \otimes \mathcal{D}x_{o,t}$$

Efficient Gradient Projection (LoGra)

- **Observation:** Gradient *DW* in backpropagation is Kronecker-product (or 2D) structured:



$$\mathrm{vec}(\mathcal{D}W) = \sum_{t=1}^{T} x_{i,t} \otimes \mathcal{D}x_{o,t}$$

  - Impose a Kronecker-product structure on the projection matrix *P*

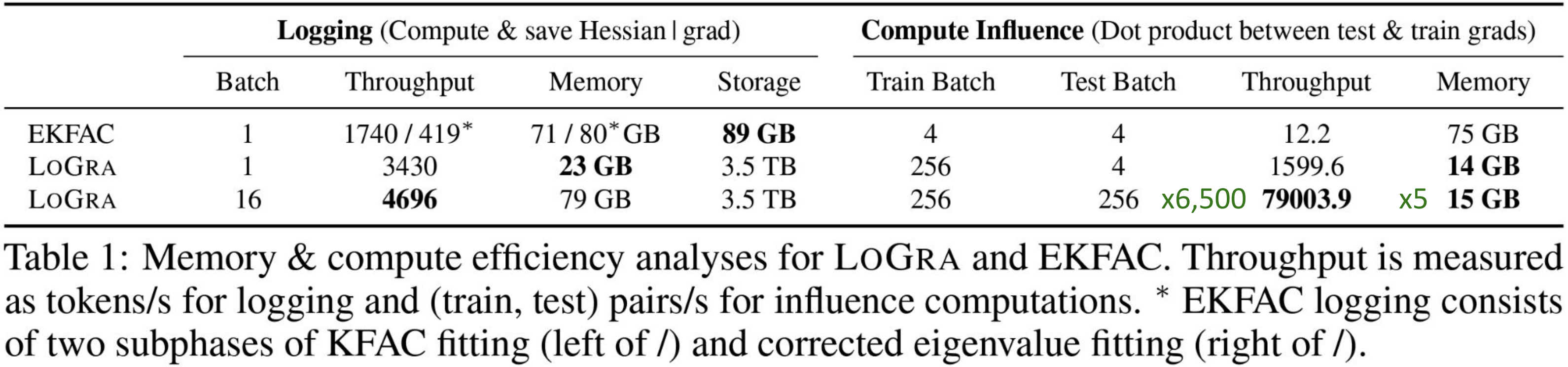
$$\boxed{P\mathrm{vec}(\mathcal{D}W)} \triangleq \boxed{(P_i \otimes P_o)}\mathrm{vec}(\mathcal{D}W) = \sum_{t=1}^{T}(P_i \otimes P_o)(x_{i,t} \otimes \mathcal{D}x_{o,t}) = \sum_{t=1}^{T}\boxed{P_i x_{i,t} \otimes P_o \mathcal{D}x_{o,t}}$$

  - Time & space complexity: $O(\sqrt{km})$ (1GB for the above example)
  - Compute projected gradients *without* materializing full gradients
  - Easy and efficient implementation with small add-on layers

Efficient Gradient Projection (LoGra)

- Vector database consists of two phases
  - Vector logging/caching
  - Vector similarity search

- Efficiency comparison with Llama3-8B-Instruct and the 1B-token dataset
  - Baseline: EKFAC

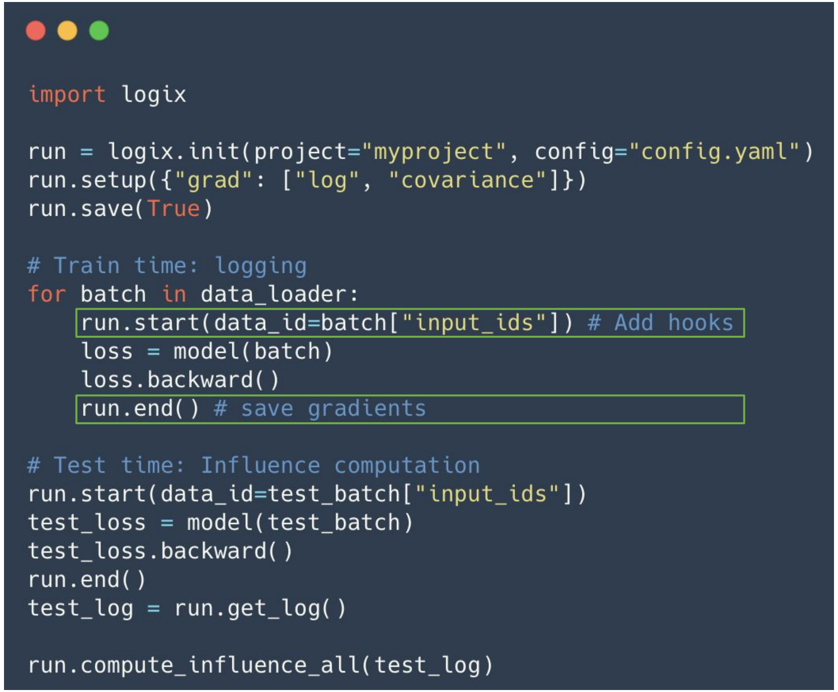


	Logging (Compute & save Hessian | grad)				**Compute Influence** (Dot product between test & train grads)			
	Batch	Throughput	Memory	Storage	Train Batch	Test Batch	Throughput	Memory
EKFAC	1	1740 / 419*	71 / 80* GB	**89 GB**	4	4	12.2	75 GB
LOGRA	1	3430	**23 GB**	3.5 TB	256	4	1599.6	**14 GB**
LOGRA	16	**4696**	79 GB	3.5 TB	256	256	x6,500 **79003.9**	x5 **15 GB**

Table 1: Memory & compute efficiency analyses for LOGRA and EKFAC. Throughput is measured as tokens/s for logging and (train, test) pairs/s for influence computations. * EKFAC logging consists of two subphases of KFAC fitting (left of /) and corrected eigenvalue fitting (right of /).

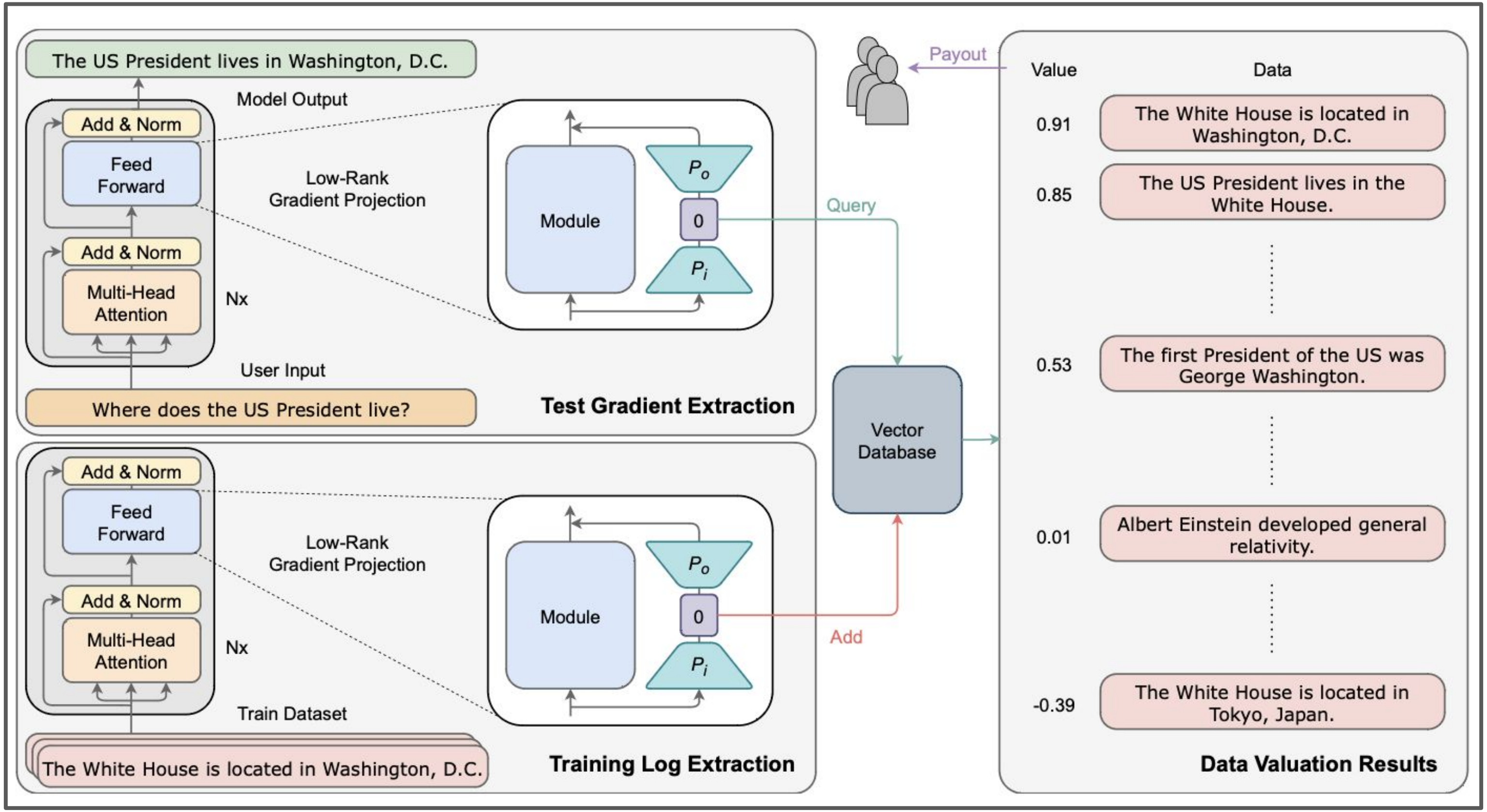
Software: LogIX

- Gradients are by-products of training
- Given arbitrary training codes,
  - Intercept gradients
  - Compute statistics (e.g., covariance)
  - Write gradients to disk
- Similar to Weights & Biases
- Compatible with various frameworks, tools, features in the LLM ecosystem!
  - Integration with HF Transformers and PyTorch Lightning



```python
import logix

run = logix.init(project="myproject", config="config.yaml")
run.setup({"grad": ["log", "covariance"]})
run.save(True)

# Train time: logging
for batch in data_loader:
    run.start(data_id=batch["input_ids"]) # Add hooks
    loss = model(batch)
    loss.backward()
    run.end() # save gradients

# Test time: Influence computation
run.start(data_id=test_batch["input_ids"])
test_loss = model(test_batch)
test_loss.backward()
run.end()
test_log = run.get_log()

run.compute_influence_all(test_log)
```

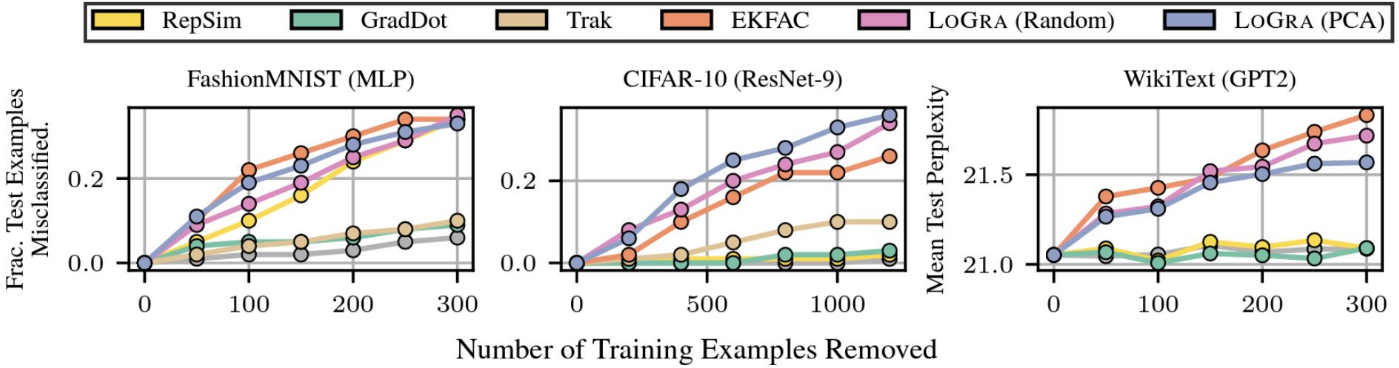
Overall TDA System Architecture



Value	Data
0.91	The White House is located in Washington, D.C.
0.85	The US President lives in the White House.
0.53	The first President of the US was George Washington.
0.01	Albert Einstein developed general relativity.
-0.39	The White House is located in Tokyo, Japan.

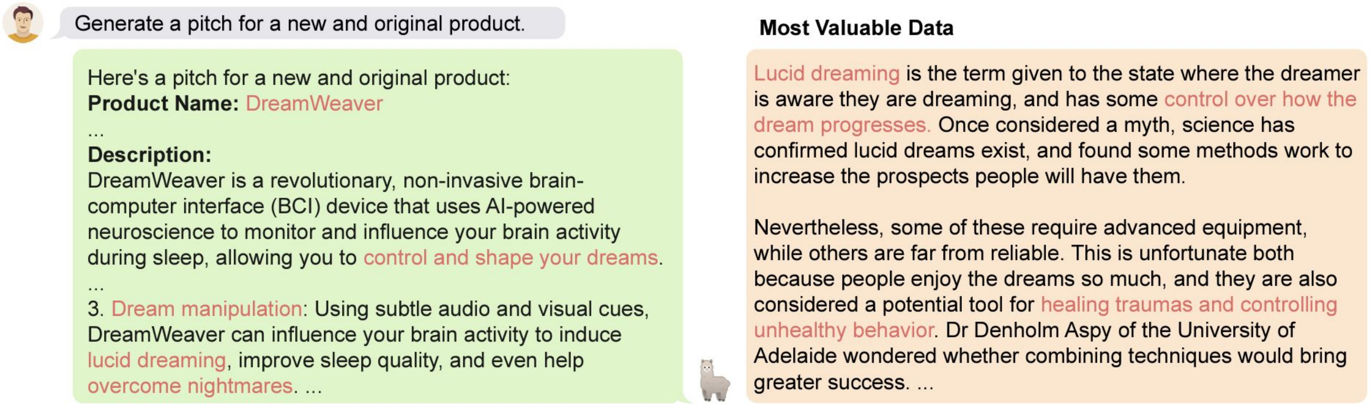
Application: Data Curation

- **Brittleness test**
  - Given a specific test data point,
  - Identify *top-k* influential training data
  - Retrain the model without the identified top-k influential data
  - Measure how much the output for this test data point changes



(a) Brittleness test

Application: Data Attribution



Generate a pitch for a new and original product.

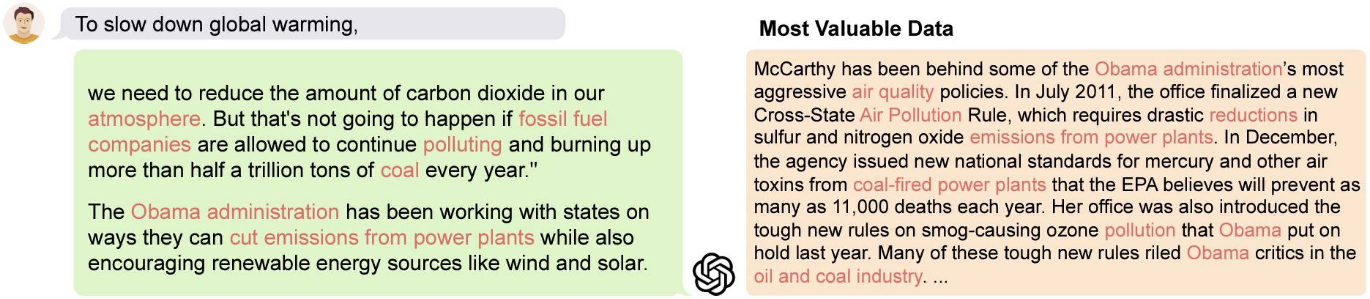
Here's a pitch for a new and original product:
Product Name: DreamWeaver
...
Description:
DreamWeaver is a revolutionary, non-invasive brain-computer interface (BCI) device that uses AI-powered neuroscience to monitor and influence your brain activity during sleep, allowing you to control and shape your dreams.
...
3. Dream manipulation: Using subtle audio and visual cues, DreamWeaver can influence your brain activity to induce lucid dreaming, improve sleep quality, and even help overcome nightmares. ...

Most Valuable Data

Lucid dreaming is the term given to the state where the dreamer is aware they are dreaming, and has some control over how the dream progresses. Once considered a myth, science has confirmed lucid dreams exist, and found some methods work to increase the prospects people will have them.

Nevertheless, some of these require advanced equipment, while others are far from reliable. This is unfortunate both because people enjoy the dreams so much, and they are also considered a potential tool for healing traumas and controlling unhealthy behavior. Dr Denholm Aspy of the University of Adelaide wondered whether combining techniques would bring greater success. ...

(a) Llama3-8B-Instruct



To slow down global warming,

we need to reduce the amount of carbon dioxide in our atmosphere. But that's not going to happen if fossil fuel companies are allowed to continue polluting and burning up more than half a trillion tons of coal every year."

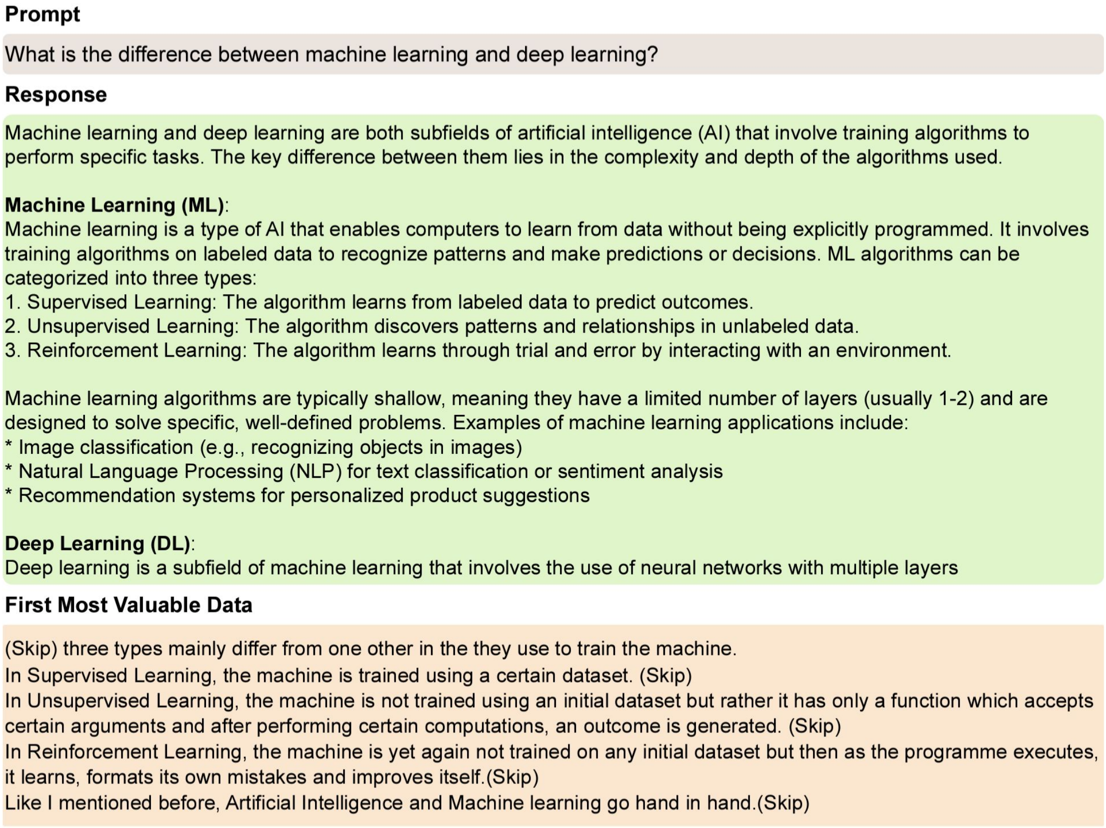
The Obama administration has been working with states on ways they can cut emissions from power plants while also encouraging renewable energy sources like wind and solar.

Most Valuable Data

McCarthy has been behind some of the Obama administration's most aggressive air quality policies. In July 2011, the office finalized a new Cross-State Air Pollution Rule, which requires drastic reductions in sulfur and nitrogen oxide emissions from power plants. In December, the agency issued new national standards for mercury and other air toxins from coal-fired power plants that the EPA believes will prevent as many as 11,000 deaths each year. Her office was also introduced the tough new rules on smog-causing ozone pollution that Obama put on hold last year. Many of these tough new rules riled Obama critics in the oil and coal industry. ...

(b) GPT2-XL (1.5B)

Application: Data Attribution



Prompt

What is the difference between machine learning and deep learning?

Response

Machine learning and deep learning are both subfields of artificial intelligence (AI) that involve training algorithms to perform specific tasks. The key difference between them lies in the complexity and depth of the algorithms used.

Machine Learning (ML):
Machine learning is a type of AI that enables computers to learn from data without being explicitly programmed. It involves training algorithms on labeled data to recognize patterns and make predictions or decisions. ML algorithms can be categorized into three types:
1. Supervised Learning: The algorithm learns from labeled data to predict outcomes.
2. Unsupervised Learning: The algorithm discovers patterns and relationships in unlabeled data.
3. Reinforcement Learning: The algorithm learns through trial and error by interacting with an environment.

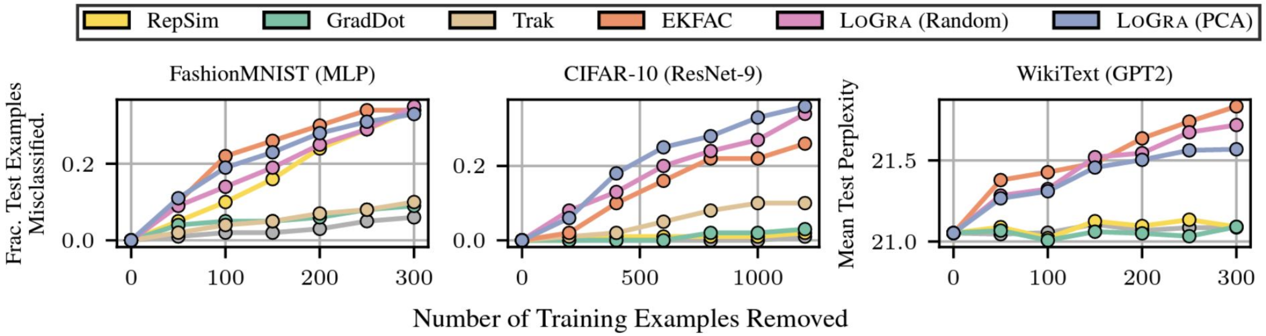
Machine learning algorithms are typically shallow, meaning they have a limited number of layers (usually 1-2) and are designed to solve specific, well-defined problems. Examples of machine learning applications include:
* Image classification (e.g., recognizing objects in images)
* Natural Language Processing (NLP) for text classification or sentiment analysis
* Recommendation systems for personalized product suggestions

Deep Learning (DL):
Deep learning is a subfield of machine learning that involves the use of neural networks with multiple layers

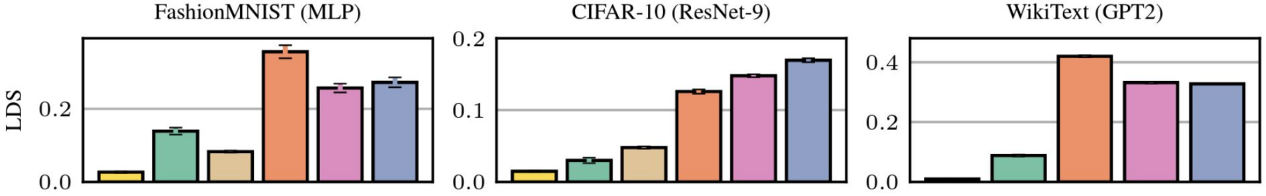
First Most Valuable Data

(Skip) three types mainly differ from one other in the they use to train the machine.
In Supervised Learning, the machine is trained using a certain dataset. (Skip)
In Unsupervised Learning, the machine is not trained using an initial dataset but rather it has only a function which accepts certain arguments and after performing certain computations, an outcome is generated. (Skip)
In Reinforcement Learning, the machine is yet again not trained on any initial dataset but then as the programme executes, it learns, formats its own mistakes and improves itself.(Skip)
Like I mentioned before, Artificial Intelligence and Machine learning go hand in hand.(Skip)

Complete Quantitative Experiments



(a) Brittleness test



(b) Linear datamodeling score (LDS)

Training Data Attribution via Approximate Unrolled Differentiation

Juhan Bae, Wu Lin, Jonathan Lorraine, Roger Grosse

Recall: Influence Functions

- Influence functions estimate the optimal solution sensitivity to downweighting a training data point.

- We discussed some conceptual issues of influence functions when applied to modern neural networks.
  - Assumes that we have found the *optimal solution* and TDA is performed on this optimal solution.
  - They cannot incorporate *the details of the training process* (e.g., the location of a data point appeared during training).

- These conceptual issues limit the applicability of influence functions. For example, we cannot perform TDA on:
  - Models not sufficiently trained near convergence.
  - Models trained with distinct stages with different objectives or datasets such as in continual learning and foundation models.

Unrolled Differentiation

- By contrast, unrolling-based TDA can incorporate details of the training process.
    - It approximates the impact of *downweighting a data point’s gradient update on the final model parameters* by backpropagating through the preceding optimization steps

- Consider optimizing the model parameters using SGD with a fixed batch size *B*:

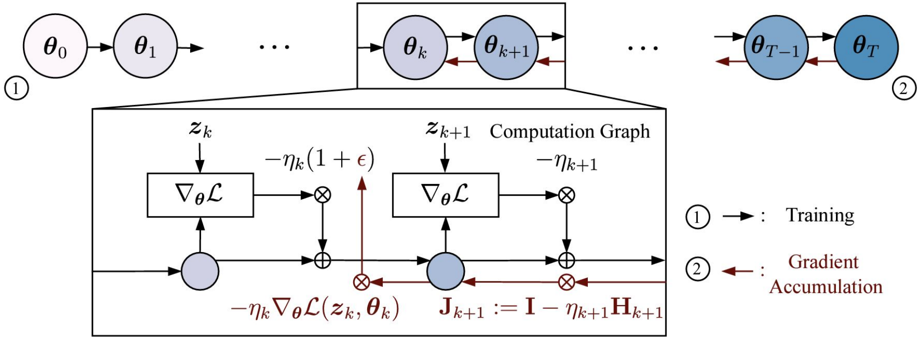
$$\boldsymbol{\theta}_{k+1} \leftarrow \boldsymbol{\theta}_k - \frac{\eta_k}{B} \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}_{ki}, \boldsymbol{\theta}_k)$$

- We aim to understand the *effect of removing a training data point*. To this end, we parameterize the update rule as:

$$\boldsymbol{\theta}_{k+1}(\epsilon) \leftarrow \boldsymbol{\theta}_k(\epsilon) - \frac{\eta_k}{B} \sum_{i=1}^{B} (1 + \delta_{ki}\epsilon)\, \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}_{ki}, \boldsymbol{\theta}_k(\epsilon)).$$

Unrolled Differentiation

- Similarly to other gradient-based TDA techniques, such as influence functions, we approximate the change in parameters with its first-order Taylor approximation.



θ_0
θ_1
θ_k
θ_{k+1}
θ_{T-1}
θ_T
①
②
z_k
z_{k+1}
Computation Graph
$-\eta_k(1+\epsilon)$
$-\eta_{k+1}$
$\nabla_\theta \mathcal{L}$
$\nabla_\theta \mathcal{L}$
$-\eta_k \nabla_\theta \mathcal{L}(z_k, \theta_k)$
$J_{k+1} := I - \eta_{k+1} H_{k+1}$
① : Training
② : Gradient Accumulation

- The total derivative at iteration *k* can be expressed as $-\frac{\eta_k}{B}\delta_k \mathbf{J}_{k+1:T}\mathbf{g}_k$, where:

$$\mathbf{J}_k := \frac{\mathrm{d}\boldsymbol{\theta}_{k+1}}{\mathrm{d}\boldsymbol{\theta}_k} = \mathbf{I} - \eta_k \mathbf{H}_k$$

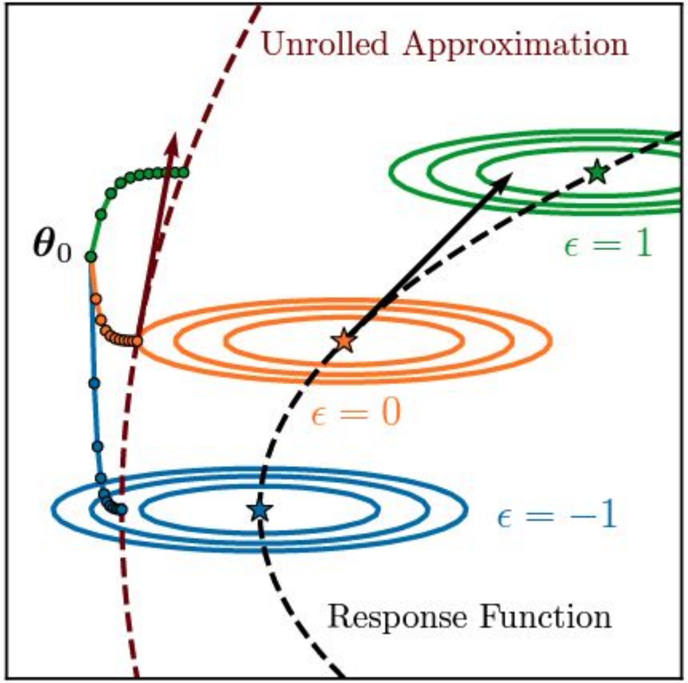
$$\mathbf{J}_{k:k'} := \frac{\mathrm{d}\boldsymbol{\theta}_{k'}}{\mathrm{d}\boldsymbol{\theta}_k} = \mathbf{J}_{k'-1} \cdots \mathbf{J}_{k+1}\mathbf{J}_k$$

$$\mathbf{g}_k := \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{z}_m, \boldsymbol{\theta}_k).$$

Unrolled Differentiation

- In contrast to influence functions, unrolling does *not assume uniqueness or convergence to the optimal solution*.

- Furthermore, it can *account for details of the training process* such as learning rate schedules, implicit bias of optimizers, or a data point’s position during training.



Unrolled Differentiation

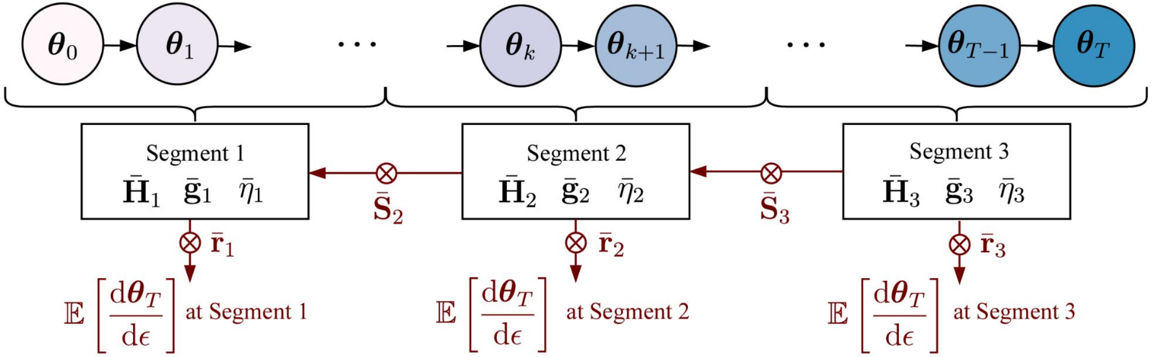
- Since the effect of removing a data point on any single training trajectory can be noisy, we consider t*he expectation over training trajectories*, where the selection of training examples in each batch are treated as random variable:

$$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right] = \mathbb{E}\left[-\sum_{k=0}^{T-1}\frac{\eta_k}{B}\delta_k \mathbf{J}_{k+1:T}\mathbf{g}_k\right] = -\sum_{k=0}^{T-1}\frac{\eta_k}{B}\mathbb{E}\left[\delta_k \mathbf{J}_{k+1:T}\mathbf{g}_k\right]$$

- **Problem:**
  - For each trajectory, the total derivative is evaluated using reverse accumulation (backpropagation) on the computation graph. However, this is prohibitively expensive, as it requires *storing all intermediate variables* for the backward pass.
  - Many Monte Carlo samples may be required to achieve accurate estimates.

SOURCE

- To derive a more efficient algorithm, we partition the training procedures into *L* segments and approximate unrolling with statistical summaries thereof.


$\boldsymbol{\theta}_0$
$\boldsymbol{\theta}_1$
$\cdots$
$\boldsymbol{\theta}_k$
$\boldsymbol{\theta}_{k+1}$
$\cdots$
$\boldsymbol{\theta}_{T-1}$
$\boldsymbol{\theta}_T$
Segment 1
$\bar{\mathbf{H}}_1$ $\bar{\mathbf{g}}_1$ $\bar{\eta}_1$
Segment 2
$\bar{\mathbf{H}}_2$ $\bar{\mathbf{g}}_2$ $\bar{\eta}_2$
Segment 3
$\bar{\mathbf{H}}_3$ $\bar{\mathbf{g}}_3$ $\bar{\eta}_3$
$\bar{\mathbf{S}}_2$
$\bar{\mathbf{S}}_3$
$\bar{\mathbf{r}}_1$
$\bar{\mathbf{r}}_2$
$\bar{\mathbf{r}}_3$
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 1
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 2
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 3

- **Key Approximations:**
  - We approximate the Hessian and gradients distribution as stationary within each segment (e.g., the Hessians within a segment share a common mean).
  - We approximate the Hessians and gradients in different time steps as statistically independent.

SOURCE

We can rewrite the previous expression with the segment notation:

$$
\begin{aligned}
\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right] &= -\mathbb{E}\left[\sum_{\ell=1}^{L}\sum_{k=T_{\ell-1}}^{T_\ell-1}\frac{\eta_k}{B}\delta_k\left(\prod_{\ell'=L}^{\ell+1}\mathbf{S}_{\ell'}\right)\mathbf{J}_{k+1:T_\ell}\mathbf{g}_k\right] \\
&= -\mathbb{E}\left[\sum_{\ell=1}^{L}\left(\prod_{\ell'=L}^{\ell+1}\mathbf{S}_{\ell'}\right)\underbrace{\left(\sum_{k=T_{\ell-1}}^{T_\ell-1}\frac{\eta_k}{B}\delta_k\mathbf{J}_{k+1:T_\ell}\mathbf{g}_k\right)}_{:=\mathbf{r}_\ell}\right] \\
&\approx -\sum_{\ell=1}^{L}\left(\prod_{\ell'=L}^{\ell+1}\underset{\textcircled{1}}{\mathbb{E}\left[\mathbf{S}_{\ell'}\right]}\right)\underset{\textcircled{2}}{\mathbb{E}\left[\mathbf{r}_\ell\right]},
\end{aligned}
$$

Note that $\mathbf{S}_\ell := \mathbf{J}_{T_{\ell-1}:T_\ell}$ is the Jacobian associated with each segment.

SOURCE - Part 1

- The first term can be approximated as follows:

$$\mathbb{E}[\mathbf{S}_\ell] = \mathbb{E}[\mathbf{J}_{T_{\ell-1}:T_\ell}] \approx \left(\mathbf{I} - \bar{\eta}_\ell \bar{\mathbf{H}}_\ell\right)^{K_\ell} \approx \exp(-\bar{\eta}_\ell K_\ell \bar{\mathbf{H}}_\ell) := \bar{\mathbf{S}}_\ell \tag{1}$$

- The above expression can be seen as applying the function $F_{\mathbf{S}}(\sigma) := \exp(-\bar{\eta}_\ell K_\ell \sigma)$ to each eigenvalue of the Hessian.

- **Intuition:**
  - The value is close to zero *in high-curvature directions*, so training “forgets” the component of the parameters which lie in these directions.
  - Information is retained throughout the segment for *low-curvature directions*.

SOURCE - Part 2

- The second term can be approximated as follows:

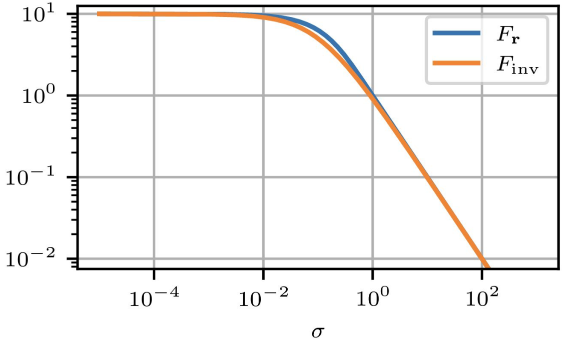
②
$$
\begin{aligned}
\mathbb{E}[\mathbf{r}_\ell] &= \mathbb{E}\left[\sum_{k=T_{\ell-1}}^{T_\ell-1} \frac{\eta_k}{B}\delta_k \mathbf{J}_{k+1:T_\ell}\mathbf{g}_k\right] \\
&\approx \frac{1}{N}\sum_{k=T_{\ell-1}}^{T_\ell-1} \bar{\eta}_\ell(\mathbf{I}-\bar{\eta}_\ell\bar{\mathbf{H}}_\ell)^{T_\ell-1-k}\bar{\mathbf{g}}_\ell \\
&= \frac{1}{N}(\mathbf{I}-(\mathbf{I}-\bar{\eta}_\ell\bar{\mathbf{H}}_\ell)^{K_\ell})\bar{\mathbf{H}}_\ell^{-1}\bar{\mathbf{g}}_\ell \\
&\approx \frac{1}{N}\underbrace{(\mathbf{I}-\exp(-\bar{\eta}_\ell K_\ell\bar{\mathbf{H}}_\ell))\bar{\mathbf{H}}_\ell^{-1}}_{:=F_{\mathbf{r}}(\sigma)}\bar{\mathbf{g}}_\ell := \bar{\mathbf{r}}_\ell,
\end{aligned}
$$

- The above expression can also be seen as applying the function $F_{\mathbf{r}}(\sigma) := \frac{1-\exp\left(-\bar{\eta}_\ell K_\ell \sigma\right)}{\sigma}$ to each eigenvalue of the Hessian.

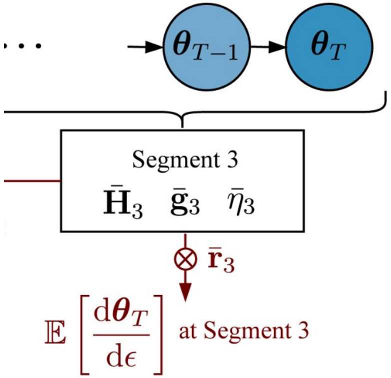
SOURCE - Part 2

$$F_{\mathbf{r}}(\sigma) \coloneqq \frac{1 - \exp\left(-\bar{\eta}_\ell K_\ell \sigma\right)}{\sigma}$$

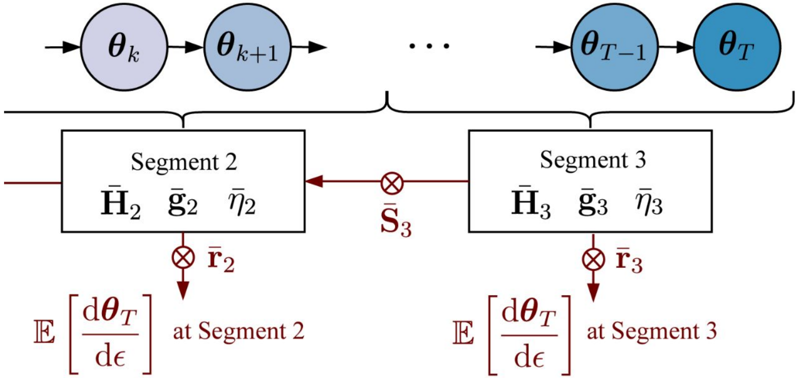
- **Intuition:** In high-curvature directions, this term approaches $^1/_\sigma$, whereas in low-curvature directions, it approaches $\bar{\eta}_\ell K_\ell$.
- Interestingly, this *qualitative behavior* can be captured with the function $F_{\text{inv}}(\sigma) \coloneqq 1/(\sigma + \lambda)$, where $\lambda = \bar{\eta}_\ell^{-1} K_\ell^{-1}$. This resembles influence function computations with *a specific damping term*: $(\bar{\mathbf{H}}_\ell + \lambda \mathbf{I})^{-1} \bar{\mathbf{g}}_\ell$



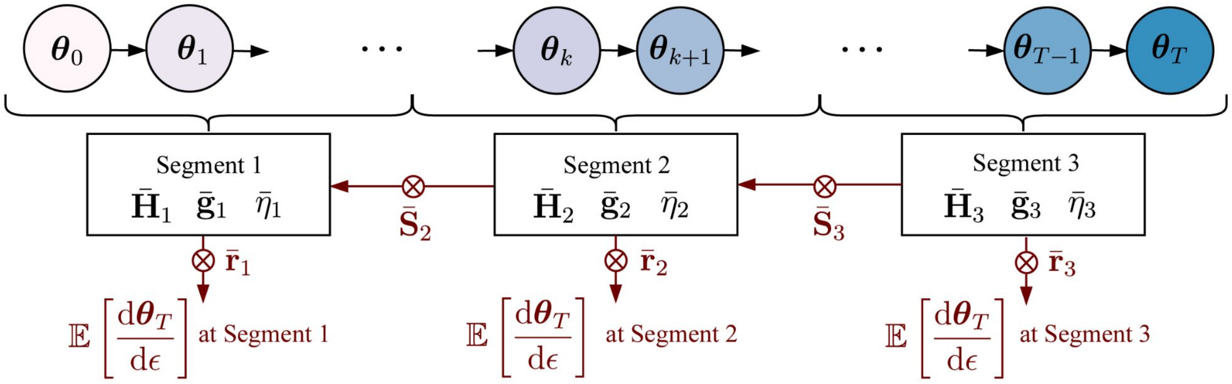
SOURCE - Full Procedure


…
$\boldsymbol{\theta}_{T-1}$
$\boldsymbol{\theta}_{T}$
Segment 3
$\bar{\mathbf{H}}_3$ $\bar{\mathbf{g}}_3$ $\bar{\eta}_3$
$\bar{\mathbf{r}}_3$
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 3

SOURCE - Full Procedure


$\boldsymbol{\theta}_k$
$\boldsymbol{\theta}_{k+1}$
$\cdots$
$\boldsymbol{\theta}_{T-1}$
$\boldsymbol{\theta}_T$
Segment 2
$\bar{\mathbf{H}}_2$ $\bar{\mathbf{g}}_2$ $\bar{\eta}_2$
$\bar{\mathbf{S}}_3$
Segment 3
$\bar{\mathbf{H}}_3$ $\bar{\mathbf{g}}_3$ $\bar{\eta}_3$
$\bar{\mathbf{r}}_2$
$\bar{\mathbf{r}}_3$
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 2
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 3

SOURCE - Full Procedure


$\boldsymbol{\theta}_0$
$\boldsymbol{\theta}_1$
$\cdots$
$\boldsymbol{\theta}_k$
$\boldsymbol{\theta}_{k+1}$
$\cdots$
$\boldsymbol{\theta}_{T-1}$
$\boldsymbol{\theta}_T$
Segment 1
$\bar{\mathbf{H}}_1$ $\bar{\mathbf{g}}_1$ $\bar{\eta}_1$
Segment 2
$\bar{\mathbf{H}}_2$ $\bar{\mathbf{g}}_2$ $\bar{\eta}_2$
Segment 3
$\bar{\mathbf{H}}_3$ $\bar{\mathbf{g}}_3$ $\bar{\eta}_3$
$\bar{\mathbf{S}}_2$
$\bar{\mathbf{S}}_3$
$\bar{\mathbf{r}}_1$
$\bar{\mathbf{r}}_2$
$\bar{\mathbf{r}}_3$
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 1
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 2
$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right]$ at Segment 3

Putting it all together, we derived a closed-form term to approximate the expected total derivative:

$$\mathbb{E}\left[\frac{\mathrm{d}\boldsymbol{\theta}_T}{\mathrm{d}\epsilon}\right] \approx -\frac{1}{N}\sum_{\ell=1}^{L}\left(\prod_{\ell'=L}^{\ell+1}\bar{\mathbf{S}}_{\ell'}\right)\bar{\mathbf{r}}_\ell$$

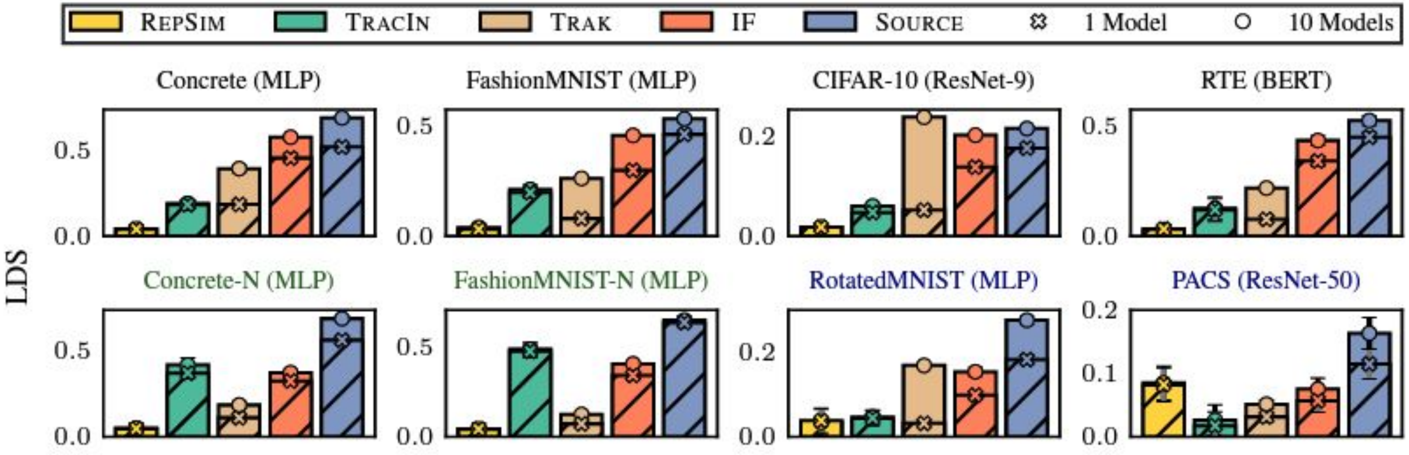
SOURCE - Implementation Details

- Given C model checkpoints saved during training,
    - SOURCE organize them into L segments. The segments may represent explicit stages of training or account for the change in Hessian and gradient throughout training.
    - Within each segment, SOURCE *estimates the stationary Hessian and gradient* by averaging them.

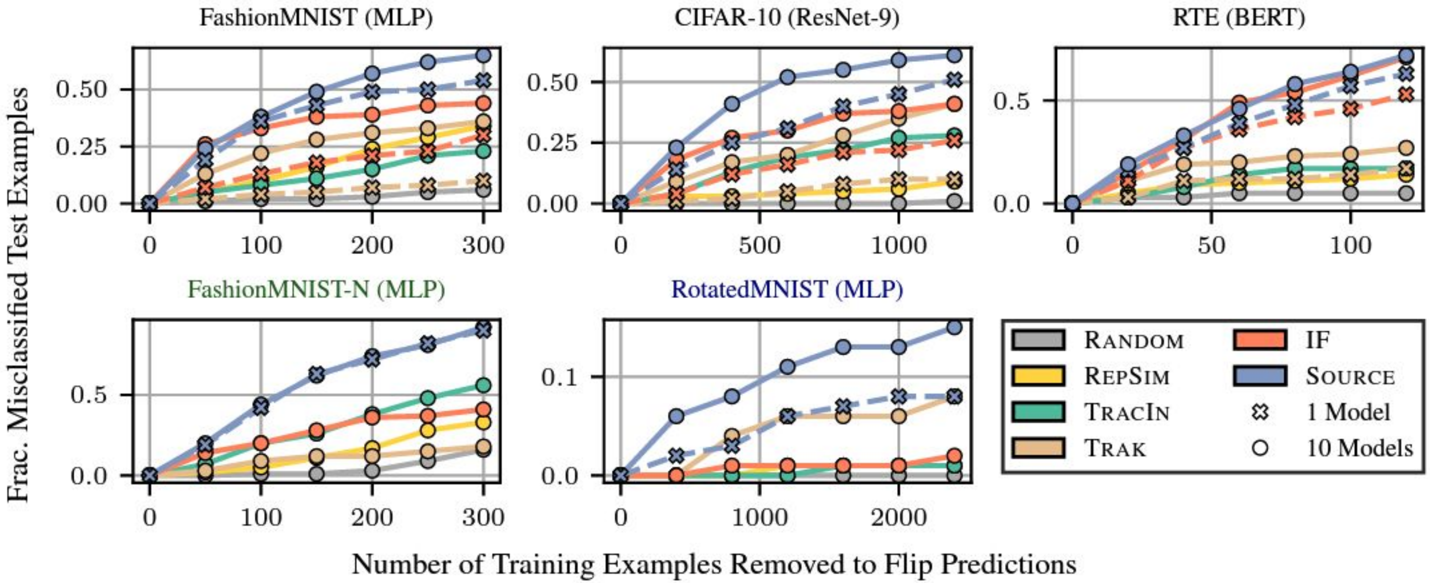
- We use EK-FAC parameterization to approximate the Hessian.
    - The EK-FAC has an explicit eigendecomposition, which enables efficient computation of the terms.

- **Computation Costs:** SOURCE requires computing the EK-FAC factors and training gradients *for each model checkpoint* when performing TDA on all segments.
    - Compared to EK-FAC IF, SOURCE is C times more computationally expensive. (We use $C = 6$ in our experiments.)
    - We do not need to save all intermediate variables for unrolled differentiation!

Linear Datamodeling Score (LDS)



Brittleness Test


FashionMNIST (MLP)
CIFAR-10 (ResNet-9)
RTE (BERT)
FashionMNIST-N (MLP)
RotatedMNIST (MLP)
Frac. Misclassified Test Examples
Number of Training Examples Removed to Flip Predictions
RANDOM
REPSIM
TRACIN
TRAK
IF
SOURCE
1 Model
10 Models