Записки научных
семинаров ПОМИ
Том 552, 2026 г.

С. И. Николенко

ФУНКЦИИ ВЛИЯНИЯ ДАННЫХ ДЛЯ БОЛЬШИХ ЯЗЫКОВЫХ МОДЕЛЕЙ

§1. Введение

Современные большие языковые модели (large language models, LLM) критически зависят от качества и состава обучающих данных. В первой части настоящего обзора мы систематически рассмотрели методы фильтрации и оценки данных, от эвристик на основе правил до сложных систем на основе LLM-судей. Эти методы полезны для удаления очевидного шума и повышения общего качества корпусов, но они не отвечают на центральный причинно-следственный вопрос: *какие конкретные обучающие примеры ответственны за заданное поведение обученной модели*?

Настоящая часть обзора посвящена *функциям влияния* (influence functions, IF) — методам, которые присваивают значения обучающим примерам на основе их контрфактического влияния на параметры и выходы модели. Центральная идея функций влияния заключается в ответе на следующий контрфактуальный вопрос: как изменилось бы поведение модели, если бы мы изменили вес конкретного обучающего примера или полностью удалили его из обучающей выборки? Этот вопрос имеет фундаментальное значение как для понимания работы моделей машинного обучения, так и для практических задач их отладки, интерпретации и улучшения.

Классический подход к ответу на этот вопрос состоял бы в том, чтобы физически удалить интересующий пример из обучающего набора и переобучить модель заново, получив так называемую оценку

Ключевые слова: функции влияния, большие языковые модели.

Работа выполнена при финансовой поддержке Министерства науки и высшего образования Российской Федерации (соглашение № 075-15-2025-344 от 29.04.2025 в Санкт-Петербургском международном математическом институте имени Леонарда Эйлера, ПОМИ РАН).

leave-one-out (LOO). Однако для современных фундаментальных моделей, обучение которых может занимать недели или месяцы на тысячах GPU и стоить миллионы долларов, такой подход является совершенно непрактичным.

Функции влияния предлагают элегантное решение этой проблемы. Вместо фактического переобучения модели они используют локальную аппроксимацию первого порядка для оценки того, как изменятся параметры модели при бесконечно малом изменении веса обучающего примера. Ключевая формула

$$I_{\text{up,params}}(z) = -H_{\widehat{\theta}}^{-1} \nabla_\theta L(z, \widehat{\theta})$$

связывает влияние примера z на параметры модели через произведение обратного гессиана $H_{\widehat{\theta}}^{-1}$ эмпирического риска и градиента функции потерь по этому примеру. Эта формула легко интерпретируется: она показывает, что влияние примера определяется не только величиной его градиента (как в наивных методах на основе функции потерь), но и направлением этого градиента относительно локальной геометрии поверхности потерь, закодированной в обратном гессиане.

Развивая эту идею, можно получить влияние на произвольную функцию от параметров модели, в частности на значение функции потерь на тестовом примере:

$$I_{\text{up,loss}}(z, z_{\text{test}}) = -\nabla_\theta L(z_{\text{test}}, \widehat{\theta})^\top H_{\widehat{\theta}}^{-1} \nabla_\theta L(z, \widehat{\theta}).$$

Эта величина тоже имеет ясную интерпретацию: если $I_{\text{up,loss}}(z, z_{\text{test}}) > 0$, то увеличение веса примера z увеличивает потери на z_{test}, следовательно z вреден для предсказания на этом тестовом примере; если же $I_{\text{up,loss}} < 0$, то z полезен. Более того, эта формула обеспечивает линейную аппроксимацию LOO-оценок: $L(z_{\text{test}}, \widehat{\theta}_{-z}) - L(z_{\text{test}}, \widehat{\theta}) \approx -\frac{1}{n} I_{\text{up,loss}}(z, z_{\text{test}})$.

Функции влияния были впервые разработаны в статистике для анализа робастности статистических оценок к выбросам и возмущениям данных [3,7]. В машинном обучении они были популяризированы в работе Koh и Liang [8], которая адаптировала классическую теорию для глубоких нейронных сетей и продемонстрировала их применимость к задачам интерпретации моделей, обнаружения ошибок в данных и исправления предсказаний. С тех пор функции влияния стали активно развивающейся областью исследований, особенно в контексте современных больших языковых моделей.

Почему функции влияния особенно важны для LLM? Большие языковые модели представляют собой новый тип для анализа влияния данных по нескольким причинам.

(1) *Масштаб*: современные LLM содержат от нескольких миллиардов до триллионов параметров и обучаются на корпусах, содержащих триллионы токенов; прямое вычисление и хранение гессианов или даже градиентов по отдельным примерам становится невозможным.
(2) *Непрозрачность*: поведение LLM часто неожиданно и труднопредсказуемо; функции влияния помогают понять, какие конкретные обучающие данные ответственны за конкретные выходы модели, что критически важно для отладки, аудита и соответствия регуляторным требованиям.
(3) *Качество данных*: обучающие корпуса LLM часто содержат шум, дубликаты, низкокачественный контент и потенциально вредоносные данные; функции влияния позволяют идентифицировать наиболее влиятельные примеры для последующей фильтрации или коррекции.
(4) *Атрибуция и ответственность*: в эпоху повышенного внимания к вопросам авторских прав, приватности и этики искусственного интеллекта возможность атрибутировать выходы модели к конкретным обучающим данным становится не просто академически интересной, но и юридически и этически важной.

Преимущества функций влияния перед альтернативными методами. В отличие от методов фильтрации, рассмотренных в первой части обзора, функции влияния обладают рядом уникальных преимуществ.

(1) *Принципиальная связь с контрфактическими оценками*: функции влияния аппроксимируют истинные LOO-оценки, позволяя предсказать, как изменится модель при удалении примера.
(2) *Учёт геометрии модели*: через обратный гессиан функции влияния учитывают локальную кривизну поверхности потерь, что позволяет отличить “сложные, но полезные” примеры от “сложных, но вредных”.

(3) *Направленный сигнал*: функции влияния измеряют не просто величину, но и направление влияния относительно конкретного тестового запроса или задачи.
(4) *Детальная атрибуция*: функции влияния позволяют проводить атрибуцию на уровне отдельных примеров и даже токенов.

Структура обзора. Настоящий обзор организован следующим образом. В разделе 2 мы подробно излагаем математическую теорию функций влияния, начиная с классического вывода теорему о неявной функции. Мы формализуем основные величины — влияние на параметры и влияние на потери, обсуждаем связь с методами ближайших соседей, рассматриваем влияние возмущений входов и объясняем адаптацию к невыпуклым задачам. Далее мы описываем практические методы вычисления обратных произведений гессиана на вектор (iHVP), включая стохастические методы типа LiSSA и факторизованные аппроксимации кривизны (K-FAC, EK-FAC).

В разделе 3 мы переходим к современным масштабируемым методам, специально разработанным для LLM. Мы подробно описываем подход “project-then-invert”, метод LoGra, который эксплуатирует послойную кронекерову структуру градиентов, и метод TRAK на основе случайных проекций.

Раздел 4 посвящён методам, которые отходят от строгой аппроксимации классических функций влияния в пользу практической масштабируемости: RapidIn — метод сжатия градиентов первого порядка, LinFIK — линеаризованное ядро будущего влияния для оценки данных на ранних стадиях обучения, и ALinFIK — его обученная аппроксимация.

В заключении мы синтезируем представленный материал, обсуждаем сильные стороны и ограничения различных подходов, даём практические рекомендации по выбору методов и намечаем направления будущих исследований.

Наша цель — предоставить самодостаточное введение в теорию и практику функций влияния для LLM, полезное как для исследователей, желающих понять математическую основу и развивать новые методы, так и для практиков, выбирающих инструменты для конкретных задач работы с данными. Мы стремимся уравновесить математическую строгость с практической применимостью, подробно объясняя как теоретические основы, так и вычислительные техники, делающие

функции влияния практически возможными в масштабе современных LLM.

§2. Функции влияния

В этом разделе мы представляем самодостаточное изложение теории функций влияния (influence functions, IF) для современных глубоких нейронных сетей и, в частности, для больших языковых моделей (large language models, LLM). Начнём с классического вывода через повышение весов обучающих примеров и формализуем основную величину

$$\mathrm{IF}(z \to v) \;=\; -\nabla_\theta v(\widehat{\theta})^\top \, H^{-1} \, \nabla_\theta \ell(z; \widehat{\theta}),$$

где H представляет собой гессиан (или его регуляризованный суррогат) целевой функции обучения в точке экстремума $\widehat{\theta}$. Затем мы опишем масштабируемые аппроксимации этой величины: стохастический оценщик на основе ряда Неймана LiSSA для вычисления произведений обратного гессиана на вектор (iHVP), подходы типа "project-then-invert", специализированные для LLM через низкоранговую структуру *градиентов* (LoGra), влияние первого порядка в стиле поиска со сжатием (RapidIn), а также линеаризованные ядра будущего влияния на ранних стадиях предобучения (LinFIK) и их обученный суррогат (ALinFIK). По ходу изложения мы определим кривизну Гаусса–Ньютона/Фишера, используемую на практике, и вспомним кронекерову структуру в градиентах по отдельным примерам, которая делает возможным подход LoGra.

2.1. Функции влияния для атрибуции данных: постановка задачи и основная идея. *Постановка задачи.* Обозначим через $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ пространство примеров, где $z = (x, y) \in \mathcal{Z}$ представляет собой размеченный пример из обучающей выборки, а $L(z, \theta)$ — дифференцируемую функцию потерь для одного примера при параметрах модели $\theta \in \Theta \subset \mathbb{R}^p$. Для обучающих точек $z_1, \ldots, z_n$ определим эмпирический риск

$$R_n(\theta) \;:=\; \frac{1}{n} \sum_{i=1}^{n} L(z_i, \theta), \qquad \widehat{\theta} \in \arg\min_\theta R_n(\theta).$$

На протяжении всего этого раздела мы предполагаем, что функция R_n дважды непрерывно дифференцируема в некоторой окрестности

точки $\widehat{\theta}$, а её гессиан

$$H_{\widehat{\theta}} \;:=\; \nabla_\theta^2 R_n(\widehat{\theta}) \;=\; \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta^2 L(z_i, \widehat{\theta})$$

является положительно определённым. Эти стандартные условия позволяют применить теорему о неявной функции и получить локальные формулы линейного отклика для изменений параметров и функции потерь при малых возмущениях обучающей выборки. Важно отметить, что данные предположения являются достаточно мягкими и выполняются для широкого класса практических задач, включая обучение глубоких нейронных сетей при условии адекватной регуляризации.

Мы следуем классическому выводу *функций влияния* (influence functions, IF), популяризированному в машинном обучении работой Koh & Liang [8]. Ключевая идея состоит в том, чтобы проследить, как бесконечно малое изменение весов отдельных обучающих примеров влияет на итоговые параметры модели и, следовательно, на её поведение на тестовых данных.

Повышение веса одного обучающего примера. Зафиксируем обучающую точку z из обучающей выборки. Рассмотрим модифицированную целевую функцию с повышенным весом этого примера:

$$R_{n,\varepsilon,z}(\theta) \;=\; R_n(\theta) + \varepsilon\, L(z,\theta), \qquad \widehat{\theta}_{\varepsilon,z} \in \arg\min_\theta R_{n,\varepsilon,z}(\theta).$$

Здесь параметр ε контролирует степень повышения веса: при $\varepsilon = 0$ мы получаем исходную задачу, а при $\varepsilon > 0$ пример z вносит дополнительный вклад в оптимизируемую функцию. Определим вспомогательную функцию $F(\theta, \varepsilon) := \nabla_\theta R_{n,\varepsilon,z}(\theta)$. Из условия оптимальности следует, что $F(\widehat{\theta}_{\varepsilon,z}, \varepsilon) = 0$ для всех значений ε.

Продифференцируем это равенство по ε в точке $\varepsilon = 0$, применяя правило дифференцирования сложной функции:

$$\underbrace{\nabla_\theta^2 R_n(\widehat{\theta})}_{H_{\widehat{\theta}}} \; \frac{d\widehat{\theta}_{\varepsilon,z}}{d\varepsilon}\Big|_{\varepsilon=0} \;+\; \underbrace{\nabla_\theta L(z,\widehat{\theta})}_{\text{прямой вклад}} \;=\; 0.$$

Решая это уравнение относительно производной параметров по ε, получаем *влияние на параметры*:

$$I_{\text{up,params}}(z) \;:=\; \frac{d\widehat{\theta}_{\varepsilon,z}}{d\varepsilon}\Big|_{\varepsilon=0} \;=\; -\,H_{\widehat{\theta}}^{-1}\,\nabla_\theta L(z,\widehat{\theta}).$$

Интуитивно говоря, эта величина представляет собой шаг метода Ньютона, который мы бы сделали, если бы вес примера z в обучающей выборке был увеличен на бесконечно малую величину. Полное удаление примера z из выборки соответствует $\varepsilon = -1/n$, что даёт линеаризованную аппроксимацию leave-one-out (LOO):

$$\widehat{\theta}_{-z} - \widehat{\theta} \;\approx\; -\frac{1}{n}\, I_{\mathrm{up,params}}(z).$$

Эти соотношения прослеживают зависимость параметров модели *непосредственно от обучающих данных* и будут служить основным инструментом для всех последующих вычислений функций влияния. Важно подчеркнуть, что данная аппроксимация является локальной и точной в первом порядке по ε; для больших возмущений могут потребоваться члены высших порядков или полное переобучение модели.

Влияние на тестовую функцию потерь. Для точки из тестового набора z_{test} определим *влияние повышения веса на потери*:

$$I_{\mathrm{up,loss}}(z, z_{\mathrm{test}}) \;:=\; \frac{d}{d\varepsilon}\, L\Big(z_{\mathrm{test}}, \widehat{\theta}_{\varepsilon,z}\Big)\,\Big|_{\varepsilon=0} \;=\; \nabla_\theta L(z_{\mathrm{test}}, \widehat{\theta})^\top\, I_{\mathrm{up,params}}(z).$$

Подставляя выражение для $I_{\mathrm{up,params}}$, получаем уже знакомую замкнутую форму:

$$I_{\mathrm{up,loss}}(z, z_{\mathrm{test}}) \;=\; -\,\nabla_\theta L(z_{\mathrm{test}}, \widehat{\theta})^\top\, H_{\widehat{\theta}}^{-1}\, \nabla_\theta L(z, \widehat{\theta}).$$

Соглашение о знаках здесь является стандартным и имеет ясную интерпретацию: если $I_{\mathrm{up,loss}}(z, z_{\mathrm{test}}) > 0$, то *увеличение* веса примера z *увеличивает* тестовые потери, следовательно, z является *вредным* для предсказания на z_{test}; если же $I_{\mathrm{up,loss}} < 0$, то z является *полезным*. Изменения при удалении одного примера (LOO) линейно аппроксимируются величиной $-\frac{1}{n} I_{\mathrm{up,loss}}(z, z_{\mathrm{test}})$. Эта формула лежит в основе многих методов интерпретации и отладки моделей машинного обучения, позволяя идентифицировать наиболее влиятельные обучающие примеры.

Возмущение обучающего входа (контрфактическое редактирование). Функции влияния также количественно описывают, как малые *редактирования* обучающего входа влияют на обученную модель. Пусть $z = (x, y)$ и $z_\delta = (x + \delta, y)$ для некоторого $\delta \in \mathbb{R}^{d_x}$. Рассмотрим замену примера z на z_δ. Используя тот же трюк с повышением веса,

но теперь с членом $+\varepsilon L(z_\delta, \theta) - \varepsilon L(z, \theta)$, получаем:

$$\frac{d\widehat{\theta}_{\varepsilon, z_\delta, -z}}{d\varepsilon}\Big|_{\varepsilon=0} \;=\; -\,H_{\widehat{\theta}}^{-1}\Big(\nabla_\theta L(z_\delta, \widehat{\theta}) - \nabla_\theta L(z, \widehat{\theta})\Big)\,.$$

Для малых δ и дифференцируемой функции потерь L выражение в скобках равно $\nabla_x \nabla_\theta L(z, \widehat{\theta})\,\delta + o(\|\delta\|)$; следовательно,

$$\widehat{\theta}_{z_\delta, -z} - \widehat{\theta} \;\approx\; -\frac{1}{n}\,H_{\widehat{\theta}}^{-1}\,\nabla_x \nabla_\theta L(z, \widehat{\theta})\,\delta.$$

Применяя правило дифференцирования сложной функции к тестовым потерям, получаем *вектор влияния возмущения входа*:

$$\begin{aligned} I_{\text{pert,loss}}(z, z_{\text{test}})^\top &:= \nabla_\delta L\Big(z_{\text{test}}, \widehat{\theta}_{z_\delta, -z}\Big)^\top\Big|_{\delta=0} \\ &= -\,\nabla_\theta L(z_{\text{test}}, \widehat{\theta})^\top\, H_{\widehat{\theta}}^{-1}\,\nabla_x \nabla_\theta L(z, \widehat{\theta}). \end{aligned}$$

Направление этого вектора указывает на то, *какие признаки* входа x наиболее сильно влияют на предсказание для z_{test} через пример z. Более того, это направление можно использовать для создания (незаметных глазу) атак на обучающую выборку, которые изменяют предсказания модели на тестовых данных. Такие атаки были продемонстрированы эмпирически в исходной работе (см. рисунок 5, стр. 6–7 в [8]), что показывает, что даже небольшие изменения обучающих данных могут иметь существенное влияние на поведение модели.

Пример логистической регрессии (почему функции влияния не эквивалентны методу ближайших соседей). Рассмотрим бинарную логистическую регрессию с метками $y \in \{-1, 1\}$ и $p(y \mid x) = \sigma(y\theta^\top x)$, где $\sigma(t) = 1/(1+e^{-t})$ — логистический сигмоид. Градиент функции потерь по одному примеру имеет вид $\nabla_\theta L(z, \theta) = -\sigma(-y\theta^\top x)\, y\, x$, а гессиан $H_{\widehat{\theta}}$ представляет собой взвешенную ковариационную матрицу:

$$H_{\widehat{\theta}} = \tfrac{1}{n}\sum_i \sigma(\widehat{\theta}^\top x_i)\sigma(-\widehat{\theta}^\top x_i) x_i x_i^\top\,.$$

Следовательно, функция влияния принимает вид

$$I_{\text{up,loss}}(z, z_{\text{test}}) \;=\; -\,y_{\text{test}} y\;\sigma(-y_{\text{test}}\widehat{\theta}^\top x_{\text{test}})\;\sigma(-y\widehat{\theta}^\top x)\;x_{\text{test}}^\top H_{\widehat{\theta}}^{-1} x.$$

По сравнению с простым скалярным произведением $x_{\text{test}}^\top x$ (которое используется в эвристиках на основе ближайших соседей), функции влияния осуществляют взвешивание с учётом:

- как обучающих, так и тестовых потерь, и

- метрики, индуцированной обратным гессианом $H_{\widehat{\theta}}^{-1}$, которая подчёркивает направления с низкой дисперсией данных.

В результате функции влияния выделяют действительно влиятельные точки, которые метод ближайших соседей может пропустить. Это происходит потому, что две точки могут быть близки в исходном пространстве признаков, но иметь совершенно разное влияние на модель в зависимости от их положения относительно границы решения и локальной кривизны функции потерь.

Данный пример наглядно демонстрирует ключевое отличие функций влияния от наивных методов поиска похожих примеров: учёт геометрии задачи оптимизации позволяет более точно идентифицировать важные обучающие примеры.

Ослабление условий выпуклости и дифференцируемости (практические замечания). Когда гессиан $H_{\widehat{\theta}}$ не является положительно определённым (например, в случае невыпуклых сетей или ранней остановки стохастического градиентного спуска), обычно добавляют тихоновскую регуляризацию $H_{\widehat{\theta}} \leftarrow H_{\widehat{\theta}} + \lambda I$ и работают с локальной выпуклой квадратичной аппроксимацией в окрестности $\widehat{\theta}$. Такой подход всё равно даёт хорошую корреляцию с изменениями при удалении примеров из выборки.

Параметр регуляризации λ обычно подбирается эмпирически, обеспечивая компромисс между точностью аппроксимации и численной устойчивостью. Типичные значения λ лежат в диапазоне от 10^{-4} до 10^{-1} в зависимости от масштаба задачи и величины собственных чисел гессиана.

Для недифференцируемых функций потерь (например, шарнирных потерь в методе опорных векторов) сглаживание восстанавливает вторые производные и позволяет вычислять предиктивные функции влияния. Например, для шарнирных потерь можно рассмотреть приближение через функцию SoftPlus:

$$\mathrm{SoftPlus}(s,t) = t \log\big(1 + \exp((1-s)/t)\big),$$

которая является гладкой и при $t \to 0$ стремится к исходным шарнирным потерям $\max(0, 1-s)$. Такое сглаживание практически не влияет на итоговые оценки влияния, но делает вычисления математически корректными и численно стабильными.

2.2. Масштабируемое вычисление $H^{-1}v$: неявные произведения гессиана на вектор и оценка LiSSA на основе ряда Неймана. Основным вычислительным узким местом при работе с функциями влияния является вычисление произведения обратного гессиана на вектор (inverse Hessian-vector product, iHVP) вида $H_{\widehat{\theta}}^{-1}v$, где $v := \nabla_\theta L(z_{\text{test}}, \widehat{\theta})$. Явное формирование матрицы гессиана $H_{\widehat{\theta}}$ требует $O(np^2)$ операций, а её обращение — $O(p^3)$ операций, что совершенно неприемлемо для современных моделей с миллионами и миллиардами параметров. Для преодоления этих вычислительных трудностей используются два стандартных подхода к ускорению [1, 6], которые мы подробно рассмотрим ниже.

Неявные произведения гессиана на вектор (HVP). Для заданного вектора v автоматическое дифференцирование позволяет вычислить отображение $u \mapsto \nabla_\theta^2 L(z_i, \widehat{\theta})\, u$ примерно с той же вычислительной стоимостью, что и вычисление градиента, используя так называемый *трюк Перлмуттера* (Pearlmutter's trick). Суммирование по мини-батчу даёт оценку $H_{\widehat{\theta}}u$ без того, чтобы явно вычислять полную матрицу $H_{\widehat{\theta}}$ в памяти. Это открывает путь к использованию методов Крылова (например, метода сопряжённых градиентов для задачи $\min_t \frac{1}{2}t^\top H_{\widehat{\theta}}t - v^\top t$) или стохастических оценщиков на основе ряда Неймана, описываемых далее.

Ключевое преимущество неявного подхода состоит в том, что мы можем работать с действием оператора $H_{\widehat{\theta}}$, не храня саму матрицу размерности $p \times p$. Это особенно критично для больших языковых моделей, где p может достигать сотен миллиардов параметров.

Стохастическая оценка LiSSA на основе ряда Неймана. Если спектральный радиус удовлетворяет условию $\rho(I - H_{\widehat{\theta}}) < 1$ (что на практике обеспечивается масштабированием или демпфированием через добавление λI), то справедливо разложение в ряд Неймана:

$$H_{\widehat{\theta}}^{-1} \;=\; \sum_{j=0}^{\infty} (I - H_{\widehat{\theta}})^j,$$

откуда

$$H_{\widehat{\theta}}^{-1}v \;=\; \lim_{J\to\infty} H_J^{-1}v, \qquad \text{где} \qquad H_J^{-1}v := \sum_{j=0}^{J} (I - H_{\widehat{\theta}})^j v.$$

Несмещённые *стохастические* аппроксимации этого ряда заменяют полный гессиан $H_{\widehat{\theta}}$ несмещённой оценкой $\widehat{H}$ на каждой итерации (например, гессианом по одному случайно выбранному обучающему примеру или мини-батчу) [1]. Удобная рекуррентная формула имеет вид:

$$\widehat{t}_0 \leftarrow v, \qquad \widehat{t}_j \leftarrow v + \bigl(I - \widehat{H}_j\bigr)\,\widehat{t}_{j-1} \quad (j = 1, \ldots, J),$$

$$\widehat{s} := \frac{1}{T}\sum_{t=1}^{T} \widehat{t}_J^{\,(t)} \;\approx\; H_{\widehat{\theta}}^{-1} v,$$

где запускается T независимых цепочек, и их результаты усредняются для снижения дисперсии оценки. Итоговая вычислительная сложность для оценки всех пар (z_i, z_{test}) относительно фиксированной тестовой точки z_{test} составляет $O(np + JT\,p)$: мы однократно вычисляем $s_{\text{test}} = \widehat{s}$, а затем для всех i находим $I_{\text{up,loss}}(z_i, z_{\text{test}}) = -s_{\text{test}}^{\top} \nabla_\theta L(z_i, \widehat{\theta})$.

Эмпирические исследования показывают, что даже небольшие значения T (порядка 5–10) дают хорошие результаты, а предсказанные изменения LOO-потерь тесно коррелируют с изменениями при фактическом переобучении модели. Это подтверждает практическую применимость линейной аппроксимации и валидирует основные теоретические предположения.

Однако существует альтернативный, детерминированный способ вычисления произведения Hv, который не требует стохастической аппроксимации и основан на структурных свойствах автоматического дифференцирования.

Трюк Перлмуттера для вычисления HVP. Основная идея состоит в вычислении *одного* произведения гессиана на вектор Hv за время и с использованием памяти, сравнимыми с вычислением одного градиента, путём специального применения автоматического дифференцирования. Этот подход является базовым строительным блоком в современных реализациях функций влияния и масштабируемых методов второго порядка.

Пусть $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ — дважды дифференцируемая функция и $\theta \in \mathbb{R}^n$. Для любого направления $v \in \mathbb{R}^n$ определим *направленную производную* (оператор R Перлмуттера, Pearlmutter's R-operator) векторнозначной функции f следующим образом:

$$R_v\{f(\theta)\} \;\stackrel{\text{def}}{=}\; \frac{d}{d\epsilon}\, f(\theta + \epsilon v)\bigg|_{\epsilon=0}. \tag{1}$$

Оператор R является линейным, удовлетворяет правилам дифференцирования произведения и сложной функции, а также коммутирует со сложением. Это делает его чрезвычайно удобным инструментом для символьных вычислений в системах автоматического дифференцирования.

Для скалярной функции потерь $\mathcal{L}$ справедливы ключевые тождества:

$$\underbrace{\nabla_\theta^2 \mathcal{L}(\theta)\, v}_{\text{произведение } Hv} \;=\; R_v\{\nabla_\theta \mathcal{L}(\theta)\} \;=\; \nabla_\theta\big(\nabla_\theta \mathcal{L}(\theta)^\top v\big). \tag{2}$$

Первое равенство непосредственно следует из определения (1), а второе вытекает из того факта, что вектор v является константой по отношению к θ. Формула (2) представляет собой алгебраическое ядро трюка Перлмуттера: произведение гессиана на вектор есть направленная производная градиента, или, эквивалентно, градиент скалярной направленной производной.

Таким образом, для вычисления Hv достаточно выполнить следующие шаги:

- вычислить градиент
$$g(\theta) = \nabla_\theta \mathcal{L}(\theta);$$
- сформировать скалярное произведение
$$s(\theta) = g(\theta)^\top v;$$
- применить обратное распространение ещё раз для получения
$$\nabla_\theta s(\theta) = Hv.$$

Современные системы автоматического дифференцирования предоставляют эту операцию напрямую, часто в виде функции `hvp(loss, theta, v)`. Суммарная стоимость составляет один прямой проход и *два* обратных прохода или, при использовании формулировки с оператором R, описанной далее, примерно стоимость вычисления одного градиента. На практике для функций влияния эти произведения HVP многократно подаются на вход процедур вычисления iHVP (таких как LiSSA или метод сопряжённых градиентов).

Рассмотрим вычислительный граф, который отображает параметры θ в скалярные потери $\mathcal{L}$ через промежуточные переменные $x_1, \dots, x_m$. Обратный режим автоматического дифференцирования (reverse-mode AD) вычисляет *сопряжённые переменные* $\bar{x}_i = \partial \mathcal{L} / \partial x_i$

путём обратного распространения чувствительностей. Идея Перлмуттера заключается в применении оператора R ко всему обратному проходу с *R-сопряжёнными переменными* $\dot{\bar{x}}_i \equiv R_v\{\bar{x}_i\}$. Используя линейность и правила дифференцирования произведения и сложной функции, получаем локальные правила обновления, например:

$$\text{если } y = f(x) \;\Rightarrow\; \bar{x} \mathrel{+}= \bar{y}\,\frac{\partial f}{\partial x}, \tag{3}$$

$$\text{то} \quad \dot{\bar{x}} \mathrel{+}= \dot{\bar{y}}\,\frac{\partial f}{\partial x} \;+\; \bar{y}\,R_v\left\{\frac{\partial f}{\partial x}\right\}. \tag{4}$$

Выполнение обычного прямого прохода (для кэширования промежуточных значений x_i), а затем одного обратного прохода, *дополненного* обновлениями через R-оператор (3), даёт в итоге $R_v\{\nabla_\theta \mathcal{L}\} = Hv$. Это в точности произведение гессиана на вектор в *точной арифметике* с точностью до машинного ϵ, без явного хранения гессиана в памяти, и с временной и пространственной сложностью, сравнимой с обратным распространением. Эта процедура “прямой-над-обратным” (forward-over-reverse) является предпочтительной версией в практических реализациях функций влияния, так как сохраняет стоимость одного HVP на уровне O(стоимость градиента).

Одно произведение гессиана на вектор по методу Перлмуттера имеет временную сложность O(стоимость градиента) и аналогичное потребление памяти; при этом не требуется хранения матриц размера $n \times n$ или $p \times p$. Именно по этой причине вычисления HVP являются основным рабочим инструментом внутри решателей для iHVP, используемых в методах функций влияния и стохастических оптимизационных алгоритмах второго порядка.

Когда вместо точного гессиана H используется обобщённая матрица Гаусса–Ньютона (Generalized Gauss–Newton, GGN) или информационная матрица Фишера (Fisher Information Matrix, FIM), соответствующее произведение на вектор допускает классическое вычисление через произведение $J^\top S J$:

$$(H_{\mathrm{GGN}} v) \;=\; J^\top\, S\,(Jv),$$

где J — это якобиан выходов сети по параметрам, а S — (положительно полуопределённый) гессиан функции потерь по этим выходам. Последовательность вычислений такова: сначала вычисляется $u = Jv$ в прямом режиме, затем поэлементно применяется Su на выходном слое,

и наконец выполняется обратное распространение $J^\top(Su)$. Эта процедура стоит примерно один прямой и один обратный проход и лежит в основе аппроксимаций натурального градиента и методов K-FAC/EK-FAC, широко используемых для эффективного обучения больших моделей.

В заключение этого раздела отметим, что комбинация трюка Перлмуттера для вычисления HVP и стохастического оценщика LiSSA для обращения гессиана позволяет масштабировать вычисление функций влияния на модели с миллиардами параметров, делая этот мощный инструмент интерпретируемости практически применимым в современном глубоком обучении.

2.3. FASTIF: практические настройки скорости и качества, распараллеливание. Метод FASTIF [6] пересматривает тот же базовый оценщик на основе LiSSA и выявляет три высокоэффективных инженерных изменения, которые сохраняют точность при значительном сокращении времени исполнения. Эти модификации показывают, что правильная инженерная реализация может дать ускорение на порядки без существенной потери в качестве атрибуции.

(1) *Предварительный отбор методом kNN*: ограничение поиска влиятельных примеров z только top-k ближайшими соседями тестовой точки z_{test} в представлениях модели (например, в пространстве финальных скрытых состояний). Это уменьшает число градиентов, которые необходимо оценить, на один-два порядка величины. Важно отметить, что показатель recall@m для действительно влиятельных точек остаётся высоким: например, $\sim$20%–60% при $k \in \{5{\times}10^3, 5{\times}10^4\}$ на корпусе размером 3.9×10^5 примеров. Это означает, что простой геометрический критерий близости в пространстве представлений оказывается хорошим предсказателем влияния, что неудивительно, учитывая локальный характер линейной аппроксимации функций влияния.

(2) *Гиперпараметры LiSSA*: использование малых размеров минибатчей ($B = 1$) с умеренной глубиной рекурсии J и компенсация дисперсии через множественные независимые запуски $T \in [1, 4]$ даёт благоприятный компромисс между скоростью и качеством (качество измеряется относительно высокоточного базового метода). Интуитивно, малый размер батча позволяет быстрее выполнять итерации, а множественные независимые

цепочки эффективно снижают стохастический шум без необходимости увеличивать J до больших значений.

(3) *Распараллеливание*: вычисление T цепочек на отдельных устройствах, однократное усреднение $\widehat{s}$ (операция all-reduce), а затем асинхронная оценка кандидатов. В экспериментальной проверке с моделью BERT-base на датасете MultiNLI эти изменения дали общее ускорение примерно в 80 раз при сохранении ранговой корреляции более 95% с полными функциями влияния. При $k = 10^3$, быстром вычислении s_{test} и использовании 4 GPU полный запрос для поиска влиятельных точек занял в этом эксперименте ~ 1.5 минуты вместо более чем 2 часов (таблица 1 и раздел 5 в работе [6]).

Эти результаты демонстрируют, что функции влияния можно сделать практически применимыми для реальных задач машинного обучения, где требуется быстрая итерация и отладка моделей.

Валидация через сравнение с leave-one-out. Для метода FASTIF были проведены две взаимодополняющие проверки [6]:

- *согласованность с полными функциями влияния*: быстрые и полные оценки влияния показывают корреляции Пирсона, Спирмена и Кендалла на уровне 99.8%/99.5%/96.9% соответственно в описанном выше эксперименте; такие высокие значения корреляций подтверждают, что аппроксимации не искажают относительный порядок влиятельных примеров;
- *фактическое переобучение*: удаление топ-m полезных примеров увеличивает потери на z_{test}, в то время как удаление топ-m вредных примеров их уменьшает; быстрые функции влияния соответствуют полным функциям влияния и существенно превосходят случайный выбор для всех значений $m \in \{1, 5, 25, 50, 100\}$.

Эти эксперименты подтверждают надёжность атрибуции данных на основе функций влияния даже при использовании сильных аппроксимаций. Более того, они демонстрируют, что теоретические гарантии локальной линейной аппроксимации работают на практике в широком диапазоне режимов удаления примеров.

2.4. Гессиан Гаусса–Ньютона и кронекеровская структура. В этом разделе мы обсудим общую математическую основу многих методов вычисления функций влияния. Дело в том, что современные

большие модели мотивируют использование *структурированных* аппроксимаций для $H_{\widehat{\theta}}$ и для градиентов по отдельным примерам. Эти структурные предположения являются ключом к масштабированию методов функций влияния на модели с миллиардами параметров.

Аппроксимации Гаусса–Ньютона и касательного пространства Фишера. Для функций потерь, основанных на правдоподобии, с выходом модели $f_\theta(x)$ и отрицательным логарифмом правдоподобия $L = -\log p_\theta(y \mid x)$, *обобщённая аппроксимация Гаусса–Ньютона* (generalized Gauss-Newton, GGN) линеаризует функцию f_θ и заменяет истинный гессиан на

$$G_{\widehat{\theta}} \approx J_{\widehat{\theta}}^\top W J_{\widehat{\theta}},$$

где $J_{\widehat{\theta}}$ представляет собой якобиан выхода f_θ по параметрам, а W — член кривизны, зависящий от выходного распределения.

Популярные схемы с факторизацией Кронекера (K-FAC/EK-FAC) [4, 10] аппроксимируют послойную матрицу Фишера или GGN с помощью факторизаций, которые дешёвы в обращении. Это позволяет реализовать масштабируемое предобуславливание второго порядка и, по тому же принципу, масштабируемые аппроксимации функций влияния на уровне токенов или отдельных примеров. Первое исследование функций влияния в масштабах LLM с использованием кривизны в стиле EK-FAC и её связи с методом Гаусса–Ньютона было представлено в работе [5].

Ключевая идея аппроксимации Гаусса–Ньютона заключается в том, что для многих функций потерь (особенно для перекрёстной энтропии) основной вклад в кривизну вносит не нелинейность модели относительно параметров, а геометрия выходного распределения. Это оправдывает замену полного гессиана на более простую и структурированную аппроксимацию GGN.

Форма Кронекера для градиентов по отдельным примерам. Рассмотрим линейный или свёрточный слой с весами $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Градиент по одному примеру имеет вид (с точностью до перестановки размерностей):

$$\nabla_W L(z, \theta) = \delta\, h^\top, \qquad \operatorname{vec}\big(\nabla_W L(z, \theta)\big) = h \otimes \delta,$$

где h — входная активация, а δ — обратно распространяемая ошибка. Эта *кронекеровская структура* лежит в основе недавно разработанных методов проекции с последующим обращением (project-then-iHVP), таких как LoGra, которые выбирают операторы проекции,

факторизующиеся вдоль прямых и обратных размерностей, и избегают вычисления полных p-мерных градиентов в памяти.

Важно подчеркнуть, что эта факторизация не является аппроксимацией — она точно отражает структуру вычислений в прямом и обратном проходах через нейронную сеть. Мы будем опираться на эту структуру в последующих разделах для построения эффективных алгоритмов атрибуции в больших масштабах.

2.5. Практическое вычисление функций влияния для z_{test}. Ниже приведён полный алгоритм вычисления функций влияния для заданной тестовой точки, объединяющий все рассмотренные выше техники:

(1) *вычислить* $v := \nabla_\theta L(z_{\text{test}}, \widehat{\theta})$ однократно (автоматическое дифференцирование); это градиент потерь на тестовой точке, который будет использоваться для всех последующих вычислений влияния;
(2) *оценить* $s_{\text{test}} \approx H_{\widehat{\theta}}^{-1} v$ одним из следующих методов:
   - (*LiSSA*) запустить T независимых рекурсий Неймана с постоянными стохастическими оценками гессиана $\widehat{H}_j$ на каждой итерации (по мини-батчу или одному примеру); усреднить T полученных векторов $\widehat{t}_J$; использовать демпфирование или масштабирование для обеспечения условия $\rho(I - \widehat{H}) < 1$, что гарантирует сходимость ряда Неймана;
   - (*FASTIF*) предпочесть малый размер батча ($B = 1$), умеренное J (выбирается на основе сходимости), и распараллелить вычисления по T независимым процессам; опционально предварительно отобрать кандидатов с помощью kNN для снижения стоимости оценки на порядок величины;
(3) *оценить всех кандидатов* z_i с помощью единственного скалярного произведения:

$$I_{\text{up,loss}}(z_i, z_{\text{test}}) \;=\; -\, s_{\text{test}}^{\top}\, \nabla_\theta L(z_i, \widehat{\theta});$$

это вычислительно простой этап: после того как s_{test} вычислен, оценка влияния каждого обучающего примера требует лишь одного скалярного произведения;
(4) *(опционально) влияние возмущения входа*: если требуется атрибуция на уровне признаков через редактирование обучающих данных, вычислить $-s_{\text{test}}^{\top} \nabla_x \nabla_\theta L(z_i, \widehat{\theta})$ для получения

$I_{\text{pert,loss}}(z_i, z_{\text{test}})$; это даст вектор чувствительности, показывающий, как изменения конкретных признаков в z_i повлияют на потери на z_{test};

(5) *проверки*: для небольшого количества наиболее полезных и наиболее вредных точек верифицировать предсказанные эффекты через фактическое переобучение с удалением m примеров (leave-m-out retraining); быстрые функции влияния обычно правильно предсказывают знаки и величины изменений.

Этот алгоритм представляет собой практически применимый рецепт для использования функций влияния в реальных задачах. Критически важным является баланс между точностью (через параметры J, T, λ) и вычислительной эффективностью (через размер батча, распараллеливание и предварительный отбор кандидатов).

2.6. Обсуждение и дополнительные ссылки. Функции влияния дают каузально-ориентированный, контрфактический ответ на вопрос “какие обучающие точки оказались важны для данного поведения модели?” с помощью замкнутых локальных линеаризаций, которые обычно весьма хорошо согласуются с переобучением leave-one-out как в выпуклых, так и в невыпуклых режимах при правильном выборе демпфирования и сглаживания.

Их основным ограничивающим фактором является вычислительная сложность: надёжная аппроксимация $H^{-1}v$ и оценка многих кандидатов требуют значительных ресурсов. Методы оценки в стиле LiSSA с использованием HVP, вместе с инженерными улучшениями типа FASTIF, делают функции влияния практичными для средних и больших моделей и датасетов, сохраняя корреляцию $> 95\%$ с полными функциями влияния при снижении временных затрат в десятки раз.

В последующих разделах мы используем рассмотренные выше структуры для масштабирования функций влияния до LLM (например в методе LoGra) и рассмотрим методы оценки данных на *ранних стадиях* обучения (LinFIK/ALinFIK). Однако базовый алгоритм, представленный выше, остаётся общим фундаментом для всех этих подходов. Понимание этих основ критически важно для правильной интерпретации результатов более сложных методов и для диагностики возможных проблем в их работе.

§3. Project-then-invert и метод LoGra

3.1. Почему сначала проекция, а затем обращение? Вспомним базовую формулу функций влияния для дважды дифференцируемого эмпирического риска $\widehat{R}(\theta) = \frac{1}{n}\sum_{i=1}^{n} L(z_i, \theta)$ в точке сходимости $\widehat{\theta}$[1]:

$$\mathrm{IF}(z_{\mathrm{tr}}, z_{\mathrm{te}}) \;=\; -\nabla_\theta L(z_{\mathrm{te}}, \widehat{\theta})^\top \, (H_{\widehat{\theta}} + \lambda I)^{-1} \, \nabla_\theta L(z_{\mathrm{tr}}, \widehat{\theta}),$$

где $H_{\widehat{\theta}}$ представляет собой гессиан (часто заменяемый на матрицу Гаусса–Ньютона или Фишера; см. раздел 3.5).

Это эквивалентно решению единственной демпфированной системы Ньютона:

$$(H_{\widehat{\theta}} + \lambda I)s = g_{\mathrm{te}} \quad \text{с} \quad g_{\mathrm{te}} = \nabla_\theta L(z_{\mathrm{te}}, \widehat{\theta}),$$

и последующему вычислению скалярного произведения $g_{\mathrm{tr}}^\top s$, где $g_{\mathrm{tr}} = \nabla_\theta L(z_{\mathrm{tr}}, \widehat{\theta})$; мы уже обсуждали и исходную формулировку, и аппроксимации на основе LiSSA.

Для современных LLM прямое вычисление или обращение $H_{\widehat{\theta}}$ невозможно из-за огромной размерности пространства (миллиарды параметров). Идея *проекции с последующим обращением* (project-then-invert) состоит в ограничении внимания на $k \ll p$ мерное подпространство, задаваемое проекцией $P \in \mathbb{R}^{k\times p}$, и вычислении влияния *внутри* этого подпространства:

$$\mathrm{IF}_P(z_{\mathrm{tr}}, z_{\mathrm{te}}) \;=\; -(Pg_{\mathrm{te}})^\top \left(P\, H_{\widehat{\theta}}\, P^\top + \lambda I_k\right)^{-1} (Pg_{\mathrm{tr}}). \tag{5}$$

Один из способов вывести формулу (5) заключается в поиске ограниченного шага Ньютона:

$$s^\star = \arg \min_{s \in \mathrm{range}(P^\top)} \frac{1}{2} s^\top (H_{\widehat{\theta}} + \lambda I)s - g_{\mathrm{te}}^\top s.$$

Записывая $s = P^\top a$, получаем систему размерности $k \times k$:

$$\left(PH_{\widehat{\theta}}P^\top + \lambda I_k\right)a \;=\; Pg_{\mathrm{te}}, \qquad s^\star \;=\; P^\top a, \qquad g_{\mathrm{tr}}^\top s^\star \;=\; (Pg_{\mathrm{tr}})^\top a.$$

Ключевое преимущество этого подхода заключается в том, что теперь нам нужно обратить только матрицу размерности $k \times k$, где k может быть выбрано достаточно малым (например, $k \sim 10^3$-10^4) для эффективного обращения, даже если исходная размерность параметров p составляет миллиарды.

---

[1]На протяжении этого раздела мы сохраняем стандартную демпфированную форму с $\lambda > 0$ как для численной устойчивости, так и для контроля направлений с малыми собственными числами.

Критический вопрос: как выбрать проекцию P, чтобы сохранить максимум информации о влиянии при минимальной размерности? Различные методы предлагают разные ответы на этот вопрос, от случайных проекций до проекций, основанных на структуре кривизны.

3.2. Метод Arnoldi-IF: подпространства Крылова для обращения. Метод Arnoldi-IF [13] строит проекцию P неявно в виде ортонормированного базиса подпространства Крылова

$$\mathcal{K}_k(H_{\widehat{\theta}}, v) = \operatorname{span}\{v, Hv, \ldots, H^{k-1}v\}$$

для начального вектора v (обычно g_{te}), используя итерации Арнольди.

Обозначим через $U \in \mathbb{R}^{p\times k}$ получившийся ортонормированный базис и через $H_k = U^\top H_{\widehat{\theta}} U$ соответствующую проекцию гессиана на это подпространство (матрица верхней хессенберговой формы). Тогда демпфированный шаг Ньютона аппроксимируется следующим образом:

$$s \;\approx\; U\,(H_k + \lambda I)^{-1}\,(U^\top g_{\text{te}}), \qquad \text{IF} \;\approx\; -\,g_{\text{tr}}^\top U\,(H_k + \lambda I)^{-1}(U^\top g_{\text{te}}).$$

Ключевое преимущество метода состоит в том, что ни полная матрица гессиана $H_{\widehat{\theta}}$, ни полноразмерные градиенты не требуют вычисления в памяти: всё, что нам необходимо, — это произведения гессиана на вектор (HVP) и произведения якобиана на вектор (JVP), которые мы вычисляем через автоматическое дифференцирование в прямом и обратном режимах.

Это сводит повторные вычисления $(H_{\widehat{\theta}} + \lambda I)^{-1} g$ к недорогим операциям в k-мерном подпространстве и позволяет масштабировать метод на трансформеры с 300 миллионами параметров и десятки или сотни миллионов примеров [13].

Интуитивно, итерация Арнольди строит базис, который хорошо приспособлен для решения линейной системы с конкретным вектором g_{te}: каждая следующая базисная функция добавляется так, чтобы лучше аппроксимировать действие обратного оператора $(H_{\widehat{\theta}} + \lambda I)^{-1}$ на исходный вектор. Это делает метод особенно эффективным, когда нам нужно решить одну или несколько похожих систем с высокой точностью.

Преимущества и недостатки. Метод Arnoldi-IF напрямую нацелен на подпространство, наиболее релевантное для линейной системы с выбранным начальным вектором (векторами), и достигает ускорения

на порядки по сравнению с iHVP-оценками типа LiSSA. Это особенно ценно в режимах, где требуется высокая точность аппроксимации обратного гессиана для конкретных тестовых примеров.

Однако у метода есть несколько ограничений:

(i) качество аппроксимации существенно зависит от выбора начального вектора (векторов); неудачный выбор начального вектора может привести к построению подпространства Крылова, которое плохо захватывает важные направления кривизны; на практике обычно используется $v = g_{\text{te}}$, но для некоторых задач может потребоваться несколько начальных векторов;

(ii) использование множественных начальных векторов увеличивает вычислительную сложность пропорционально их количеству; каждый дополнительный начальный вектор требует построения своего подпространства Крылова, что может стать узким местом при большом числе различных тестовых запросов;

(iii) построение базиса U требует выполнения многих произведений гессиана на вектор (HVP); хотя каждое отдельное HVP эффективно вычисляется через автоматическое дифференцирование, суммарная стоимость k таких операций (по одной на каждый базисный вектор) может быть значительной.

На практике метод Arnoldi-IF часто комбинируется с предварительной фильтрацией кандидатов (например, через k-NN в пространстве представлений модели) для снижения количества обучающих точек, которые необходимо оценить. Это гибридное решение объединяет сильные стороны обоих подходов: геометрическая фильтрация отсеивает явно нерелевантные примеры, а точный метод Арнольди обеспечивает высококачественное ранжирование оставшихся кандидатов.

Метод также хорошо подходит для задач, где требуется атрибуция для небольшого фиксированного набора тестовых примеров: в этом случае стоимость построения базисов Крылова амортизируется по многим обучающим примерам, которые нужно оценить.

3.3. Метод TRAK: линеаризация и случайные проекции после ядра. Метод TRAK [12] рассматривает атрибуцию через призму локальной линейной модели в окрестности финальной обученной сети. Ключевая идея состоит в том, чтобы линеаризовать классификатор (например, превратить его в логистическую регрессию последнего

слоя на замороженных признаках) и получить замкнутую форму для шага Ньютона в этой линеаризованной задаче.

Композиция этого решения со *случайной* проекцией P даёт эффективно вычислимую оценку влияния в форме (5). Метод сохраняет эмпирическую точность, близкую к дорогостоящим методам атрибуции, основанным на переобучении (datamodel attributions), оставаясь при этом вычислительно tractable в масштабах ImageNet, CLIP и BERT.

Использование случайных признаков (random features) позволяет однократно закэшировать $Pg(z)$ для всех обучающих точек и лишь решать небольшую систему размерности $k \times k$ для каждой тестовой точки. Это делает онлайн-запросы атрибуции чрезвычайно быстрыми после однократной оффлайн-фазы предобработки.

Теоретическое обоснование использования случайных проекций основано на результатах из теории случайных матриц и сжимающих отображений (compressed sensing): при достаточно большом k случайная проекция с высокой вероятностью сохраняет структуру исходного пространства градиентов. Это особенно хорошо работает в режиме, когда градиенты лежат в подпространстве меньшей размерности (что часто наблюдается на практике из-за корреляций между примерами и низкоранговой структуры весовых матриц).

Два практических замечания:

(i) линейная оценка влияния данных TRAK (Linear Datamodeling Score, LDS) оценивает качество метода атрибуции через корреляцию Спирмена между предсказанным и наблюдаемым качеством модели на тестовых данных при множественных переобучениях на половинах датасета; этот подход полезен в режимах, когда контрфактическое обучение всё ещё осуществимо в умеренных масштабах (например, для датасетов размером до нескольких миллионов примеров); LDS предоставляет количественную метрику для сравнения различных методов атрибуции: чем выше корреляция между предсказаниями метода и реальными изменениями после переобучения, тем более надёжным является метод, что особенно ценно для валидации новых подходов к атрибуции, поскольку даёт объективное измерение их качества;

(ii) ограничение влияния только последним слоем часто недостаточно для больших языковых моделей: масса атрибуции распределена по множеству слоёв, и разные части модели вносят вклад

в итоговое предсказание различными способами, поэтому выбор признаков и дизайн проекции P имеют критическое значение для успеха метода; в практических реализациях TRAK для LLM часто используются многослойные признаки или признаки из нескольких ключевых слоёв (например, промежуточные слои трансформера в дополнение к финальному), что позволяет получить более богатую картину того, как обучающие данные влияют на поведение модели на разных уровнях абстракции.

Важным преимуществом TRAK является его кросс-модальная применимость: метод успешно работает не только для текстовых моделей (BERT), но и для визуальных задач (ImageNet, CLIP) и мультимодальных архитектур. Это свидетельствует о том, что базовые принципы линеаризации и проекции являются достаточно универсальными и не зависят от специфики конкретной модальности данных.

Метод также демонстрирует хорошую устойчивость к выбору гиперпараметров: случайные проекции относительно нечувствительны к конкретной реализации, если размерность k выбрана достаточно большой. Это делает TRAK хорошим выбором для быстрого прототипирования и первичной диагностики моделей, когда требуется быстро получить разумные оценки влияния без тщательной настройки гиперпараметров.

3.4. Метод LoGra: использование кронекеровской структуры в градиентах. Метод LoGra (Low-rank Gradient structure) [2] специально разработан для масштабирования функций влияния на большие языковые модели путём эксплуатации естественной факторизации градиентов в нейронных сетях.

Кронекеровская структура в градиентах. Рассмотрим линейный или свёрточный слой с входными активациями $x_i \in \mathbb{R}^{n_i \times T}$, выходом $x_o = W x_i$, и длиной последовательности T. Обратное распространение даёт:

$$\operatorname{vec}(\mathrm{D}W) \;=\; \sum_{t=1}^{T} x_{i,t} \otimes \mathrm{D}x_{o,t}. \tag{6}$$

Выберем проекцию с кронекеровской структурой:

$$P \;=\; P_i \otimes P_o, \qquad P_i \in \mathbb{R}^{k_i \times n_i}, \quad P_o \in \mathbb{R}^{k_o \times n_o}, \quad k_i k_o = k_{\text{layer}}.$$

Тогда проекция градиента принимает факторизованный вид:

$$P\,\mathrm{vec}(\mathrm{D}W) \;=\; \sum_{t=1}^{T}(P_i x_{i,t}) \otimes (P_o \mathrm{D}x_{o,t}). \tag{7}$$

Иначе говоря, в этом случае мы

- проецируем как *прямые* активации, так и *обратные* чувствительности независимо, а также
- восстанавливаем спроецированный градиент напрямую из этих низкоразмерных потоков.

Это снижает стоимость проекции для каждого батча с $O(bkp)$ до $O\big(b\sqrt{kp}\big)$ и избегает хранения полных градиентов по отдельным примерам, что приводит к высокой утилизации GPU и низкому потреблению памяти в масштабах миллиардов параметров. Квадратный корень в сложности возникает из-за того, что $k = k_i \cdot k_o$, и мы обрабатываем каждую размерность независимо.

Полный алгоритм LoGra. Метод LoGra состоит из двух фаз.

(1) *Однократно (оффлайн)*: для каждого обучающего примера z запустить модель с небольшими *дополнительными* модулями проекции для записи $Pg(z)$ на диск. Эти модули просто применяют матрицы P_i и P_o к потокам активаций и градиентов соответственно.
(2) *Во время атрибуции (онлайн)*: вычислить Pg_{te} с теми же модулями, построить матрицу $k \times k$ вида $PH_{\widehat{\theta}}P^{\top}$ (в проекции), и использовать формулу (5) для вычисления влияния.

Спроецированные градиенты затем сохраняются в векторной базе данных; многие системы сводят задачу атрибуции к поиску по сходству плюс решение небольшой линейной системы. На модели Llama3-8B с датасетом размером 1 миллиард токенов LoGra показывает пропускную способность до 6,500× выше и использование памяти GPU в 5 раз ниже по сравнению с методом на основе EK-FAC, который был единственной альтернативой, работающей в таких масштабах на момент публикации [2].

Демпфирование как спектральное разреживание. Почему такая проекция имеет смысл, и какую P выбрать? Диагонализация гессиана $H_{\widehat{\theta}} = Q\Lambda Q^{\top}$ показывает, что демпфированное обращение умножает собственные компоненты на $(\lambda_j + \lambda)^{-1}$: направления с малыми собственными числами подавляются. Если мы (мягко) отбрасываем

направления с низкой кривизной, мы получаем *спектральный разреживатель градиента*.

Метод LoGra формализует эту идею и мотивирует выбор P для сохранения направлений с высокой дисперсией в потоке градиентов. Это приводит к простой инициализации через PCA для прямых и обратных потоков каждого слоя — эмпирически такой подход превосходит случайную проекцию P. Интуитивно говоря, мы хотим сохранить те направления, вдоль которых градиенты варьируются больше всего, так как именно эти направления вносят основной вклад в изменение параметров и, следовательно, в функции влияния.

3.5. От Гаусса–Ньютона/Фишера к K-FAC/EK-FAC (и обратно к LoGra). Для функций потерь на основе перекрёстной энтропии с softmax (обобщённый) метод Гаусса–Ньютона (GNH) совпадает с информационной матрицей Фишера (FIM) $F = \mathbb{E}\,[J^\top \mathrm{Cov}_y J]$, где J — якобиан логитов по параметрам.

Метод K-FAC аппроксимирует эту кривизну послойно через произведение Кронекера:

$$F_{\text{layer}} \;\approx\; A \otimes G,$$

где A представляет собой второй момент входов, а G — второй момент выходных градиентов. Это даёт эффективное обращение и предобуславливание натуральным градиентом [10].

Метод EK-FAC дополнительно диагонализует блок с факторизацией Кронекера в его собственном базисе для лучшей аппроксимации истинной кривизны [4].

Эти методы дают две полезные идеи в нашем контексте:

(i) они оправдывают использование GNH/Фишера вместо $H_{\widehat{\theta}}$ в формуле (5), и
(ii) они подсказывают использование *структурированных* послойных собственных направлений для матрицы P (например, можно инициализировать P_i, P_o через ведущие собственные подпространства матриц A, G).

Важно отметить, что эти аппроксимации не только вычислительно эффективны, но и имеют теоретическое обоснование: для широкого класса функций потерь GNH и матрица Фишера обеспечивают лучшую локальную квадратичную аппроксимацию, чем полный гессиан, особенно в режимах, где модель близка к интерполяции данных.

3.6. Набросок алгоритма атрибуции методом LoGra. Полный алгоритм использования LoGra для атрибуции данных состоит из следующих шагов:

(1) *однократно (оффлайн):*
- для каждого слоя зафиксировать матрицы проекции P_i, P_o (через PCA или на основе информации о кривизне);
- для каждого обучающего примера z выполнить стандартный прямой и обратный проход, дополненный вычислением по формуле (7) для получения $Pg(z)$;
- одновременно накапливать спроецированную кривизну $C := PH_{\hat{\theta}}P^\top$ (оценка GNH/Fisher);
- если обучающий корпус слишком велик, этот подход можно комбинировать с векторной базой данных в стиле FAISS для предварительной фильтрации кандидатов перед решением системы размерности $k \times k$; это особенно важно для LLM, где обучающие датасеты могут содержать триллионы токенов;

(2) *атрибуция (онлайн):* для данной тестовой точки z_{te}
- вычислить Pg_{te} с теми же модулями проекции,
- решить систему $(C + \lambda I)a = Pg_{\text{te}}$,
- выдать оценки влияния $s(z) = -(Pg(z))^\top a$ для всех обучающих примеров;
- опционально можно также выдать топ-K наиболее влиятельных примеров и/или суммы влияний по группам (например, по документам или темам);

(3) *сложность:* вычисление спроецированного градиента имеет сложность $O(b\sqrt{kp})$ на батч, где b — размер батча; онлайн-решение системы требует $O(k^3)$ операций на тестовую точку (обычно пренебрежимо мало при $k \sim 10^3$-10^4); требующийся объём памяти для хранения масштабируется как $O(k\,|D|)$, где $|D|$ — размер датасета.

Эта структура алгоритма делает возможной атрибуцию в реальном времени даже для моделей с десятками миллиардов параметров, обученных на триллионах токенов.

3.7. Сравнение семейства методов проекции с последующим обращением. Различные методы из семейства project-then-invert различаются в основном способом построения проекции P и обработки спроецированной кривизны.

Метод	Как строится P	Как обрабатывается $PH_{\hat{\theta}}P^\top$	Примечания
Arnoldi-IF [13]	Базис Крылова из HVP	Решение малой системы $(H_k + \lambda I)$	Сильное обращение без iHVP; чувствительно к начальному вектору.
TRAK [12]	Случайные признаки / линеаризация	Малая система на тестовую точку	Отличная практичность для разных модальностей.
LoGra [2]	Послойные $P_i \otimes P_o$	Точная проекция матрицы GNH / Фишера	Проекция за $O(\sqrt{kp})$; пропускная способность в масштабах LLM.

Каждый из этих методов имеет свои преимущества в зависимости от размера модели, доступных вычислительных ресурсов и требований к точности атрибуции.

3.8. Практические рекомендации и режимы отказа. *Выбор размерности k.* Лучше начать с бюджета, дающего $k \in [10^3, 10^4]$ суммарно по всем слоям; его можно распределить пропорционально числу параметров или объяснённой дисперсии. Большее значение k помогает больше всего, когда предварительная фильтрация кандидатов слаба. В практических экспериментах обычно наблюдается насыщение качества атрибуции при $k > 10^4$, что делает дальнейшее увеличение k неэффективным.

Выбор демпфирования λ. Можно рассматривать λ как спектральный фильтр; наилучшее значение можно найти поиском по логарифмической сетке, предпочитая значения, которые стабилизируют решение систем $k \times k$ и снижают чувствительность топ-K результатов к выбору k. Типичные значения лежат в диапазоне $\lambda \in [10^{-4}, 10^{-1}]$, но оптимальное значение сильно зависит от конкретной задачи и модели.

Какую кривизну использовать? Для классификационных LLM оценка GNH/Фишера обычно работает хорошо и доступна *бесплатно* один раз за шаг во многих фреймворках. Обновления в стиле EK-FAC (редкое обновление собственных векторов, частое отслеживание собственных чисел) могут улучшить стабильность, если поддерживать $PH_{\hat{\theta}}P^\top$ онлайн.

Проверки качества. Когда возможно переобучение в малых масштабах, можно использовать анализ хрупкости (удалить топ-K примеров

и переобучить) для проверки качества атрибуции. В масштабах LLM это сложно, но можно попробовать проверку с отложенными примерами, для которых наиболее влиятельные обучающие фрагменты известны заранее (например, шаблонные факты или специально внедрённые примеры).

Фильтрация кандидатов. Даже с быстрой проекцией оценка *всех* обучающих точек затратна. Один из вариантов — предварительно отобрать 10^3-10^4 кандидатов на тестовую точку через kNN в пространстве представлений перед применением формулы (5). Метод FASTIF показывает, что это сохраняет большую часть истинных топ-K при ускорении всего процесса в 50-80 раз. Однако следует быть осторожным: слишком агрессивная предварительная фильтрация может пропустить действительно влиятельные, но геометрически далёкие примеры.

3.9. Связи с классическими выводами функций влияния. Стохастическая оценка iHVP на основе ряда Неймана (LiSSA) остаётся надёжным базовым методом, когда n мало или когда точные HVP дешёвы в вычислении. Однако для высокой точности могут потребоваться тысячи итераций. Методы проекции с последующим обращением избегают этой проблемы по построению, решая напрямую систему меньшей размерности.

Наконец, отметим, что все методы в стиле функций влияния полагаются на локальную квадратичную аппроксимацию в окрестности $\widehat{\theta}$. Для невыпуклых LLM мы полагаемся на демпфирование, замену на GNH или матрицу Фишера и эмпирическую валидацию. На практике аппроксимации, описанные в разделах 3.2–3.4, остаются информативными и применимыми, даже когда строгие теоретические предположения нарушены.

Это подчёркивает важность эмпирической валидации: теоретические гарантии дают нам понимание того, почему методы работают, но окончательное подтверждение их применимости должно происходить через тщательную проверку на реальных задачах с известными свойствами (например, через контролируемые эксперименты с искусственно внедрёнными влиятельными примерами).

§4. Методы RapidIn, LinFIK и ALinFIK

4.1. RapidIn: оценка влияния первого порядка со сжатием градиентов. *Мотивация и постановка задачи.* Пусть $\mathcal{D} = \{s_i\}_{i=1}^N$ представляет собой множество обучающих последовательностей, где каждая s_i состоит из промпта и сгенерированной целевой последовательности. Обозначим через t тестовый промпт с соответствующей генерацией, произведённой текущей моделью.[2] Пусть θ — параметры модели, а $L(\cdot, \theta)$ — функция потерь (на токен или на последовательность) при обучении.

Следуя стандартным идеям влияния первого порядка (как в методе TracIn), эффект обновления параметров θ на тестовую последовательность t при выполнении одного шага градиентного спуска по обучающей последовательности s_a на итерации a можно получить из разложения Тейлора в окрестности θ_a:

$$L(t, \theta_{a+1}) = L(t, \theta_a) + (\theta_{a+1} - \theta_a)^\top \nabla_\theta L(t, \theta_a) + \underbrace{\mathcal{O}\big(\|\theta_{a+1} - \theta_a\|^2\big)}_{\text{игнорируется для малых шагов}}.$$

При обновлении градиентным спуском,

$$\theta_{a+1} = \theta_a - \eta_a \nabla_\theta L(s_a, \theta_a),$$

и малой скорости обучения η_a из формулы (4.1) следует влияние первого порядка последовательности s_a на t на итерации a:

$$L(t, \theta_a) - L(t, \theta_{a+1}) \;\approx\; \eta_a \, \nabla_\theta L(s_a, \theta_a)^\top \nabla_\theta L(t, \theta_a). \tag{8}$$

Это выражение имеет ясную интерпретацию: изменение потерь на тестовом примере пропорционально скалярному произведению градиентов по обучающему и тестовому примерам. Чем больше градиенты выровнены по направлению, тем сильнее обучение на s_a уменьшает потери на t.

Суммируя по всем итерациям, где появляется s_k (или усредняя по батчу при $b > 1$), получаем полное влияние обучающей последовательности s_k на генерацию t (первого порядка):

$$I_\theta(t, s_k) = \sum_{a:\, s_a = s_k} \eta_a \nabla_\theta L(s_k, \theta_a)^\top \nabla_\theta L(t, \theta_a) \approx e\eta \nabla_\theta L(s_k, \theta)^\top \nabla_\theta L(t, \theta), \tag{9}$$

[2] Мы используем те же обозначения и разбиение последовательностей на токены, что и в оригинальной работе RapidIn [9].

где e — число эпох обучения, и мы использовали тот факт, что для типичных режимов дообучения LLM скорость обучения и параметры меняются медленно, так что градиенты можно вычислить в некоторой точке θ. Эта аппроксимация особенно точна для поздних стадий обучения, когда модель приближается к локальному минимуму и траектория параметров стабилизируется.

Поскольку как выходы LLM, так и обучающие примеры являются последовательностями токенов, метод RapidIn также определяет влияния на уровне токенов для различных комбинаций:

$$I_\theta(t, s_k^i) = e\eta \sum_j \nabla_\theta L(s_k^i, \theta)^\top \nabla_\theta L(t^j, \theta), \tag{10}$$

$$I_\theta(t^j, s_k) = e\eta \sum_i \nabla_\theta L(s_k^i, \theta)^\top \nabla_\theta L(t^j, \theta), \tag{11}$$

$$I_\theta(t^j, s_k^i) = e\eta \, \nabla_\theta L(s_k^i, \theta)^\top \nabla_\theta L(t^j, \theta), \tag{12}$$

где верхний индекс обозначает конкретный токен в последовательности. Эти метрики полезны для детального анализа ошибок и атрибуции на уровне отдельных токенов, что особенно важно для отладки языковых моделей и понимания того, какие части обучающих данных повлияли на порождение конкретных слов или фраз.

Почему необходимо сжатие. Прямое вычисление формул (9)–(12) требует хранения или повторного вычисления градиентов по отдельным примерам $\nabla_\theta L(s_k, \theta)$, размерность которых равна числу параметров модели (десятки миллиардов и более для современных LLM). Даже кэширование одного градиента на обучающий пример исчерпает терабайты памяти и сделает поиск запретительно медленным.[3]

Это фундаментальное ограничение памяти делает наивный подход к вычислению функций влияния первого порядка непрактичным для современных LLM. Метод RapidIn решает эту проблему путём сжатия каждого градиента по примеру в вектор *RapidGrad* фиксированной малой длины K (размером в килобайты или единицы мегабайт), что позволяет быстро вычислять скалярные произведения в сжатом пространстве [9]. Ключевая идея состоит в том, что для оценки влияния нам не нужны точные градиенты — достаточно сохранить структуру их скалярных произведений.

[3] В работе RapidIn сообщается, что один градиент полной точности для модели LLaMA-2 7B занимает ∼26 ГБ; жёсткий диск на 10 ТБ вместит только ∼393 таких градиента [9].

Фаза кэширования RapidIn — построение RapidGrad. Для каждой обучающей последовательности s_k выполняются следующие шаги.

(1) Вычислить градиент по примеру $g_k = \nabla_\theta L(s_k, \theta)$ и применить *послойную* ℓ_2-нормализацию; конкатенировать/выровнять слои в вектор $v \in \mathbb{R}^p$. Послойная нормализация смягчает хрупкость, возникающую из-за различий в масштабах между слоями в глубоких сетях. Это критически важно, поскольку без нормализации градиенты последних слоев могут доминировать в сжатом представлении, что приведёт к потере информации о влиянии ранних слоёв.
(2) *Случайное перемешивание*: применить λ раундов перестановок и перестановок строк/столбцов для декорреляции локальных паттернов перед применением скетчинга (Алгоритм 1 в оригинальной работе [9]). Этот шаг необходим, потому что последовательные параметры в нейронных сетях часто имеют коррелированные градиенты, и прямое применение блочного суммирования может привести к потере информации из-за деструктивной интерференции.
(3) *Случайная проекция / скетчинг*: сгенерировать вектор Радемахера $\rho \in \{-1, +1\}^p$, сформировать поэлементное произведение $v \odot \rho$, и суммировать непересекающиеся блоки размера p/K для получения $r_k \in \mathbb{R}^K$: $(r_k)_j = \sum_{i=(j-1)p/K+1}^{jp/K} (v \odot \rho)_i$. Это сжатие в стиле count-sketch / случайной проекции, которое приближённо сохраняет скалярные произведения между векторами с высокой вероятностью. Теоретическое обоснование этого подхода основано на результатах из теории случайных матриц, показывающих, что при достаточном K искажение скалярных произведений ограничено с высокой вероятностью.
(4) *Кэширование* r_k (в половинной точности) на диске, в памяти CPU или GPU для последующего использования при поиске. Использование половинной точности (float16) дополнительно снижает требования к памяти в два раза без существенной потери точности для большинства приложений.

Важно отметить, что эта процедура сжатия выполняется *однократно* для всех обучающих примеров и может быть тривиально распараллелена, что делает её применимой даже для очень больших датасетов.

Фаза поиска RapidIn — быстрая оценка влияния. Для данного тестового порождения t вычисляется его градиент $g_t = \nabla_\theta L(t, \theta)$, который затем сжимается в r_t с использованием того же конвейера обработки, что и для обучающих примеров. Затем влияние аппроксимируется следующим образом:

$$I_\theta(t, s_k) \;\approx\; e\eta\, r_k^\top r_t, \qquad I_\theta(t^j, s_k) \approx e\eta \sum_i r_k (s^i)^\top r_t(t^j), \text{ и т.д.} \tag{13}$$

где для оценок на уровне токенов сжатие применяется к градиентам по отдельным токенам перед агрегацией, как в формулах (10)–(12).

Критически важно, что и фаза кэширования, и фаза поиска тривиально распараллеливаются по GPU и процессам, что позволяет эффективно использовать современное высокопроизводительное оборудование. После того как все RapidGrad векторы вычислены, задача поиска влиятельных примеров сводится к задаче поиска ближайших соседей в пространстве $\mathbb{R}^K$, для которой существуют высокоэффективные специализированные алгоритмы и библиотеки (например, FAISS).

Эмпирическая эффективность. Метод RapidIn демонстрирует сжатие размера градиента более чем в $> 2{\times}10^5$ раз (например, до $K = 2^{16}$, что составляет $\sim$125 КБ) и ускорение поиска на порядки величины. Для моделей семейства LLaMA-2 оценка влияния для всего обучающего набора из 52 тысяч примеров может быть выполнена за минуты после того, как RapidGrad векторы закэшированы. Это представляет собой радикальное улучшение по сравнению с наивным подходом, который был бы абсолютно неприменим на практике.

Версия RapidIn на уровне токенов улучшает анализ ошибок и устойчивости по сравнению с оценкой на уровне последовательностей. Эксперименты с бэкдорами и возмущениями демонстрируют высокую точность поиска "отравленных" или возмущённых примеров, а также преимущества оценки на уровне токенов [9]. Это особенно важно для задач безопасности и надёжности LLM, где необходимо идентифицировать специфические токены или фразы в обучающих данных, которые могут вызывать нежелательное поведение модели.

Связь с функциями влияния. Формула (9) представляет собой *предобусловленный единичной матрицей* частный случай функций влияния (замена H^{-1} на I), мотивированный динамикой первого порядка

на ранних шагах обучения и выбранный из соображений масштабируемости. На практике RapidIn жертвует некоторой точностью по сравнению с точными функциями влияния ради массивного выигрыша в применимости к LLM с миллиардами параметров.

Эта аппроксимация оправдана наблюдением, что для многих практических задач атрибуции геометрия скалярных произведений градиентов уже содержит большую часть релевантной информации, и учёт кривизны через гессиан даёт лишь небольшое улучшение, не оправдывающее многократного увеличения вычислительной сложности.

4.2. Линеаризованное ядро будущего влияния (LinFIK) и его обучаемая аппроксимация (ALinFIK). *Постановка задачи.* Метод LinFIK [11] решает задачу более раннего этапа: *оценку будущей ценности данных на ранних стадиях обучения*, например на шаге t, до полного завершения обучения до состояния θ_T. Метод отвечает на вопрос: какие примеры x_i максимально улучшат производительность на тестовом наборе $Y = \{y_j\}_{j=1}^{M}$, если мы продолжим обучение?

Ключевая идея заключается в линеаризации одного шага обучения в точке t и использовании динамики первого порядка для определения ядра, которое оценивает обучающие точки по тому, насколько они, как ожидается, уменьшат потери на Y на следующем шаге. Это особенно полезно в сценариях, где необходимо принять решение о составе обучающей выборки до завершения дорогостоящего процесса обучения большой модели.

От одного шага обучения к ядру. Пусть w_t — параметры на шаге t, а $w_{t+1} = w_t - \eta_t \nabla_w L(x_i, w_t)$ — обновление после обучения на примере x_i со скоростью обучения η_t. Тогда *мгновенное* изменение тестовых потерь на примере y составляет

$$\begin{aligned} K_{w_t}(y, x_i) &:= L(y, w_t) - L(y, w_{t+1}) = (w_t - w_{t+1})^\top \nabla_w L(y, w_t) \\ &= \eta_t \, \nabla_w L(x_i, w_t)^\top \nabla_w L(y, w_t), \end{aligned} \tag{14}$$

где мы пренебрегли членами порядка $\mathcal{O}(\eta_t^2)$. Это выражение количественно описывает, насколько один градиентный шаг по примеру x_i уменьшает потери на тестовом примере y.

Усредняя по тестовому набору Y (специфичному для задачи), получаем *линеаризованное ядро будущего влияния* (linearized future

influence kernel, LinFIK):

$$\begin{aligned}\mathrm{LinFIK}(Y, x_i) &:= \frac{1}{M}\sum_{j=1}^{M} K_{w_t}(y_j, x_i) \\ &= \frac{\eta_t}{M}\sum_{j=1}^{M} \nabla_w L(x_i, w_t)^\top \nabla_w L(y_j, w_t).\end{aligned} \tag{15}$$

Таким образом, LinFIK представляет собой скалярное произведение между градиентом (на ранней стадии обучения) примера x_i и *агрегированным* градиентом валидационного набора Y в той же контрольной точке. Это даёт эффективный способ предсказать, какие обучающие примеры будут наиболее полезны для улучшения производительности на конкретном тестовом распределении, без необходимости полного завершения обучения.

Почему достаточно градиентов на ранних шагах: стабильность. Работа [11] обосновывает использование формулы (15) на ранних шагах, показывая, что градиенты по отдельным примерам меняются *медленно* по направлению и сохраняют порядок норм в процессе обучения LLM. Эти наблюдения имеют фундаментальное значение для практической применимости метода.

Запишем разложение первого порядка для поля градиентов между моментами t и T:

$$g_T(x) = \nabla_w L(x, w_T) \;=\; g_t(x) + H\Delta w + \mathcal{O}\big(\|H\Delta w\|^2\big)\,,$$

где $H = \nabla_w^2 L(\cdot, w_t)$ — гессиан, а $\Delta w = w_T - w_t$ — изменение параметров, получаем два ключевых свойства:

- *стабильность направления*: косинус угла между градиентами $\cos(g_t, g_T) = \dfrac{g_t^\top g_T}{\|g_t\|\|g_T\|} \approx 1$ при достаточно мягких предположениях, то есть градиенты быстро выравниваются по мере обучения и затем сохраняют своё направление; это означает, что относительная важность различных примеров, измеренная через выравнивание градиентов, устанавливается рано и остаётся стабильной;
- *стабильность порядка норм*: если $\|g_t(x_i)\| > \|g_t(x_j)\|$, то с высокой вероятностью $\|g_T(x_i)\| > \|g_T(x_j)\|$; другими словами, примеры с большими ранними градиентами, как правило, сохраняют большие градиенты и на поздних стадиях обучения;

это свойство позволяет идентифицировать “трудные” или влиятельные примеры на ранних этапах.

Эти наблюдения, которые авторы называют феноменом “медленного изменения” или “ленивого обновления градиентов” (lazy gradient update), поддерживают использование скалярных произведений на ранних шагах как надёжных предсказателей будущего влияния. Авторы также представляют эмпирические визуализации на наборе моделей Pythia, которые подтверждают оба наблюдения [11]. Эти визуализации показывают, что корреляция между ранними и поздними градиентами остаётся высокой на протяжении всего обучения, что валидирует основную гипотезу метода.

Практическое вычисление. Вычисление по формуле (15) всё ещё требует множества градиентов по отдельным примерам. Поэтому реализация повторно использует конвейер сжатия градиентов из RapidIn (см. §4.1) для кэширования сжатых градиентов и эффективного вычисления LinFIK на практике. Это демонстрирует синергию между различными методами в экосистеме атрибуции данных: техники сжатия, разработанные для одной задачи, оказываются полезными и для других.

От LinFIK к ALinFIK (дистиллированная модель предсказания). Даже при использовании сжатия вычисление по формуле (15) для *всех* примеров x_i и большого набора Y может быть ресурсозатратным, особенно когда требуется многократная переоценка при изменении состава тестового набора или целевого распределения. Метод ALinFIK (Approximated LinFIK) предлагает *обучить* лёгкий предсказатель оценок LinFIK, действуя по следующей процедуре:

(1) *подвыборка*: выбрать небольшое множество $X_s \subset \mathcal{D}$ примеров из обучающего датасета; размер этого множества обычно составляет лишь небольшую долю (например, 1-10%) от полного датасета, что делает последующие вычисления практически возможными;
(2) *вычисление*: получить оценки метода LinFIK как оракула, обозначаемые как $S_{\mathrm{LinFIK}}(x)$, для всех $x \in X_s$ в ранней контрольной точке (используя сжатые градиенты); эти оценки рассматриваются как “истинные” метки влияния для обучения предсказательной модели;
(3) *обучение*: обучить компактную модель (например, BERT-base с ~ 110 миллионами параметров) для регрессии отображения $x \mapsto \widehat{\mathrm{LinFIK}}(x)$ на парах данных $(X_s, S_{\mathrm{LinFIK}})$; важно, что эта модель

не обязана воспроизводить лингвистические способности большой LLM — ей нужно лишь научиться распознавать паттерны в данных, коррелирующие с их будущей полезностью;

(4) *предсказание*: использовать обученную модель для предсказания оценок ALinFIK $\widehat{\text{LinFIK}}(x)$ для всех примеров $x \in \mathcal{D}$ и выполнить ранжирование/отбор на основе этих оценок.

Алгоритм 1 в работе [11] представляет этот алгоритм в точной форме. Ключевая идея подхода заключается в том, что маленькой модели не нужно воспроизводить лингвистическую компетентность LLM; ей нужно лишь обучиться *сигналу предпочтения данных*, закодированному в LinFIK. Это также служит мотивацией для выбора относительно скромной модели BERT-base в качестве предсказателя: её мощности достаточно для обучения паттернов влияния, но она остаётся вычислительно дешёвой.

Такой подход к дистилляции оценок влияния представляет собой мета-обучение: мы обучаем малую модель предсказывать, какие данные будут полезны для обучения большой модели, не требуя от малой модели самой решать целевую задачу.

Эмпирические свойства и системный взгляд. В экспериментах на модифицированных датасетах Alpaca и WikiText метод ALinFIK достигает или превосходит альтернативные подходы при огромной разнице в эффективности. Для модели размером 0.5 миллиарда параметров по сравнению с LoGra метод ALinFIK снижает использование памяти GPU в 15 раз, дискового пространства в 398 раз, а времени выполнения — в 594 раза. По сравнению с RapidIn снижения составляют $5\times$, $19\times$ и $32\times$ соответственно.

Более того, методы LoGra и RapidIn исчерпывают память (падают с OOM) при масштабировании до моделей размером 8 миллиардов параметров, в то время как ALinFIK продолжает функционировать. Это демонстрирует качественное преимущество подхода, основанного на обучении предсказателя: он позволяет разделить сложность вычисления оракульных оценок (которая выполняется лишь для подвыборки) и сложность применения к полному датасету (которая определяется размером малой модели-предсказателя).

Работа также помещает ALinFIK в контекст системы оценки данных, где предсказанные оценки влияния служат как для отбора данных, так и для их ценообразования [11]. Это открывает интересные перспективы для рынков данных, где провайдеры могут оценивать

ценность своих данных для конкретных задач обучения до фактической продажи или использования.

Важные замечания. LinFIK представляет собой принципиальную *одношаговую* линеаризацию обучения, то есть влияние примера x_i в момент t пропорционально скалярному произведению градиентов на ранних шагах; это можно сравнить с формулой RapidIn (9). Метод *не* включает предобуславливание гессианом, как классические функции влияния вида $g_t^\top H^{-1} g_i$, но обосновывается наблюдаемой стабильностью градиентов и необходимостью эффективного прогнозирования *будущего* влияния.

Фактически можно рассматривать LinFIK как упреждающий (proactive) метод атрибуции, в отличие от ретроспективных (retrospective) методов вроде классических функций влияния, которые объясняют, почему модель ведёт себя определённым образом после завершения обучения. LinFIK же предсказывает, какие данные будут полезны до завершения обучения, что делает его подходящим для задач фильтрации и выбора данных.

На практике авторы также используют сжатие в стиле RapidIn для вычисления оракульных меток, используемых при обучении предсказателя ALinFIK, что демонстрирует композиционность различных техник в этой области.

Как и в случае любой системы оценки данных, существуют ограничения области применимости (например, другие языки, кроме английского, или соображения приватности и ценообразования), которые обсуждаются авторами. Важно понимать, что методы, основанные на градиентах на ранних стадиях обучения, могут не учитывать феномены, возникающие только на поздних стадиях (например, внезапное появление новых способностей), хотя эмпирические данные показывают, что для большинства практических задач эти методы работают хорошо.

4.3. Обсуждение и выбор методов атрибуции данных. *Что именно мы атрибутируем?* Поскольку функция потерь L суммирует потери по токенам, функции влияния на уровне последовательностей атрибутируют *все* решения по токенам совместно. Если вместо этого требуется атрибуция на уровне отдельных токенов, следует определить $v_t(\theta) = \ell_t(z_{\mathrm{te}}, \theta)$ для потерь по токену t и вычислить $\mathrm{IF}(z \to v_t)$. При этом влияние на уровне последовательности равно сумме влияний по всем токенам.

Это различие важно для интерпретации результатов: атрибуция на уровне последовательности может скрывать тот факт, что различные части обучающего примера влияют на различные части порождения. Атрибуция на уровне токенов обеспечивает более детальной понимание, но требует больших вычислительных затрат и может быть более зашумленной.

Ограничения методов функций влияния. Оценки влияния являются приближениями первого порядка — нелинейные взаимодействия между обучающими точками не захватываются этими методами. Например, два примера по отдельности могут иметь малое влияние, но их совместное присутствие в обучающей выборке может давать синергетический эффект, который линейные аппроксимации не обнаружат.

Оценки могут быть хрупкими к неточной оценке кривизны и к выбросам в нормах градиентов. Демпфирование и нормализация помогают на практике, но не являются панацеей. В частности, примеры с очень большими нормами градиентов могут получать непропорционально большие оценки влияния, даже если их фактический вклад в обучение ограничен из-за регуляризации или механизмов стабилизации обучения.

Важно также понимать, что функции влияния измеряют лишь *локальное* влияние в окрестности текущих параметров. При больших изменениях данных или параметров локальные аппроксимации могут существенно отличаться от истинных эффектов. Поэтому валидация через реальное переобучение остаётся золотым стандартом для проверки предсказаний функций влияния.

Когда предпочесть какой инструмент? Выбор метода атрибуции зависит от конкретных требований задачи, доступных вычислительных ресурсов и стадии жизненного цикла модели.

- Если вы можете позволить себе оценку кривизны и хотите максимального соответствия классическим функциям влияния и методам вроде LiSSA, то методы в стиле EK-FAC обеспечивают высокое качество, но могут быть вычислительно затратными и требовать много памяти в масштабах LLM. Этот подход рекомендуется для критически важных приложений, где точность атрибуции имеет первостепенное значение.
- Если требуется очень высокая пропускная способность при обработке множества тестовых запросов, можно использовать

методы проекции градиентов (TRAK/LoGra), которые превращают задачу атрибуции данных в векторный поиск. Структурированная проекция метода LoGra предлагает дополнительную экономию, эксплуатируя послойную кронекеровскую структуру. Эти методы особенно хороши для интерактивных систем, где пользователи могут запрашивать атрибуцию для различных тестовых примеров в режиме реального времени.

- Если вам нужно *выбрать* обучающие данные до полного завершения обучения (например, для фильтрации датасета или для рынков данных), можно использовать ядра на ранних стадиях обучения (LinFIK) или их обученные аппроксимации (ALinFIK), которые работают на параметрах w_t ранних контрольных точек. Это позволяет принимать решения о составе данных на основе предсказаний их будущей ценности, что может существенно сократить затраты на эксперименты с обучением.
- В любом случае, RapidIn и его трюк с RapidGrad являются отличными инструментами для снижения размерности при небольшой потере производительности и точности относительно функций влияния. Этот метод можно рассматривать как универсальный ускоритель, который может комбинироваться с другими подходами для улучшения их практической применимости.

Дополнительно стоит отметить, что в практических системах часто используются гибридные подходы: например, грубая предварительная фильтрация с помощью быстрых методов первого порядка (RapidIn) с последующим уточнением результатов более точными, но дорогими методами (функции влияния с точным гессианом) для топ-кандидатов. Такая многоуровневая стратегия позволяет сбалансировать точность и эффективность.

§5. Заключение

В настоящем обзоре мы представили систематическое изложение теории и практики функций влияния как метода атрибуции данных для больших языковых моделей. Начав с классического вывода через теорему о неявной функции, мы проследили развитие методов от точных, но вычислительно затратных формулировок до масштабируемых аппроксимаций, применимых к моделям с миллиардами параметров.

5.1. Основные выводы. *Функции влияния — единственный контрфактический метод.* В отличие от альтернативных методов фильтрации и оценки данных, рассмотренных в первой части обзора, функции влияния обеспечивают принципиальную связь с контрфактическими оценками: они аппроксимируют, как изменится модель при удалении или изменении обучающего примера, не требуя фактического переобучения. Эта связь с leave-one-out оценками делает функции влияния золотым стандартом для задач, где важна точная атрибуция ответственности за поведение модели — например, для аудита, отладки или соответствия регуляторным требованиям. Более того, через обратный гессиан функции влияния учитывают геометрию пространства параметров, позволяя отличить примеры, градиенты которых выровнены с желаемым направлением обучения, от примеров с большими градиентами, но в нежелательных направлениях.

Масштабируемость достигается через аппроксимации. Прямое применение классической формулы функций влияния к современным LLM невозможно из-за огромной размерности пространства параметров. Однако за последние годы были разработаны эффективные аппроксимации, использующие различные компромиссы между точностью и вычислительной эффективностью. Итеративные методы типа LiSSA обеспечивают хорошие аппроксимации без явного формирования гессиана. Факторизованные аппроксимации кривизны (K-FAC, EK-FAC) эксплуатируют кронекерову структуру для эффективного представления и обращения. Методы проекции градиентов (TRAK, LoGra) превращают задачу атрибуции в векторный поиск, достигая высокой пропускной способности. Методы первого порядка (RapidIn) и методы раннего прогнозирования (LinFIK/ALinFIK) отходят от строгой аппроксимации классических функций влияния, но обеспечивают огромные выигрыши в масштабируемости при сохранении практической полезности. Важно, что эти методы не исключают друг друга: в практических системах часто используются гибридные подходы, комбинирующие быструю предварительную фильтрацию с точной атрибуцией для отобранных кандидатов.

Разные задачи требуют разных методов. Выбор конкретного метода атрибуции данных должен определяться требованиями задачи. Для ретроспективной атрибуции после обучения — когда нужно понять, какие обучающие данные ответственны за конкретный выход обученной модели — предпочтительны методы, аппроксимирующие классические

функции влияния с учётом кривизны (EK-FAC) или, при необходимости высокой пропускной способности, методы проекции градиентов (LoGra). Для выбора данных до завершения обучения — когда нужно предсказать, какие данные будут полезны в будущем — подходят методы раннего прогнозирования (LinFIK/ALinFIK), которые могут оценивать данные на основе градиентов ранних итераций. Для интерактивных систем с многочисленными запросами атрибуции критична скорость, и здесь методы сжатия (RapidIn) или проекции (LoGra) превращают задачу в быстрый векторный поиск. Для задач, где точность атрибуции менее критична, чем масштабируемость, подходы первого порядка обеспечивают приемлемое качество при драматическом сокращении вычислительных затрат.

5.2. Практические рекомендации. На основе анализа различных методов мы можем дать следующие рекомендации для практических применений.

(1) *Для критически важных применений*, где точность атрибуции имеет принципиальное значение, лучше использовать наиболее точные методы (EK-FAC) и валидировать их предсказания через выборочное фактическое переобучение.
(2) *Для высокой пропускной способности* при обработке множества тестовых запросов можно использовать методы проекции градиентов (TRAK/LoGra), которые превращают атрибуцию в векторный поиск.
(3) *Для выбора обучающих данных* до полного завершения обучения подходят ядра на ранних стадиях (LinFIK) или их обученные аппроксимации (ALinFIK).
(4) *В общем случае* рекомендуется комбинировать методы фильтрации из первой части обзора с функциями влияния: грубая фильтрация на основе правил, затем быстрая предварительная оценка методами первого порядка (RapidIn), и наконец точная атрибуция для лучших кандидатов методами с учётом кривизны.

5.3. Ограничения и открытые вопросы. Несмотря на значительный прогресс в теории и практике функций влияния для LLM, остаётся ряд важных ограничений и открытых вопросов.

Локальность аппроксимаций. Все методы функций влияния основаны на локальных аппроксимациях первого или второго порядка в

окрестности текущих параметров модели. Это означает, что они точно описывают влияние малых изменений данных, но могут существенно ошибаться при больших возмущениях. В частности, при удалении значительной доли обучающих данных или при существенных изменениях их состава поверхность потерь может измениться так, что локальные аппроксимации перестанут быть валидными. Разработка методов, которые могли бы надёжно предсказывать эффекты больших изменений данных, остаётся нерешённой задачей.

Взаимодействия между примерами. Текущие методы функций влияния рассматривают влияние каждого обучающего примера независимо, игнорируя нелинейные взаимодействия между примерами. На практике же примеры могут иметь синергетические или антагонистические эффекты: два примера по отдельности могут иметь малое влияние, но их совместное присутствие может существенно изменить поведение модели (например, через создание "пути обучения" от одного концепта к другому). Методы, основанные на теории множеств Шепли, пытаются учитывать такие взаимодействия, но их вычислительная сложность делает их непрактичными для LLM. Разработка масштабируемых методов для оценки взаимодействий высокого порядка остаётся важной задачей.

Нестационарность обучения. Методы раннего прогнозирования (LinFIK/ALinFIK) предполагают, что градиенты меняются медленно в процессе обучения, что позволяет предсказывать будущую полезность данных на основе ранних итераций. Однако это предположение может нарушаться в некоторых режимах обучения, особенно при фазовых переходах или при внезапном появлении новых способностей. Понимание того, когда и почему предположение о стабильности градиентов справедливо, требует дальнейшего теоретического и эмпирического анализа.

Интерпретируемость для многозадачных моделей. Современные LLM обучаются на разнообразных данных для выполнения множества задач. Одни и те же обучающие данные могут положительно влиять на одни способности модели и отрицательно на другие. Текущие методы функций влияния оценивают влияние относительно конкретной тестовой функции потерь или запроса, но не предоставляют целостной

картины влияния на весь спектр способностей модели. Разработка методов для многомерной атрибуции влияния, которые могли бы одновременно оценивать влияние данных на различные аспекты поведения модели, является важным направлением будущих исследований.

Вычислительные затраты для очень больших моделей. Несмотря на значительный прогресс в масштабируемости, даже самые эффективные методы (RapidIn, ALinFIK) сталкиваются с ограничениями при масштабировании до моделей с сотнями миллиардов или триллионами параметров. Хранение даже сжатых представлений градиентов для миллиардов обучающих примеров требует терабайтов памяти. Дальнейшее развитие методов сжатия, квантизации и иерархической обработки необходимо для обеспечения практической применимости к самым крупным моделям следующего поколения.

Валидация и метрики качества. Оценка качества методов функций влияния остаётся сложной задачей. Идеальным золотым стандартом были бы истинные LOO-оценки, полученные через фактическое переобучение, но они недоступны в практически интересных масштабах. Существующие метрики (например, корреляция с LOO на малых моделях, способность обнаруживать намеренно испорченные данные) являются лишь прокси-метриками для истинного качества атрибуции. Разработка более надёжных и практически применимых метрик для валидации методов атрибуции данных является важной методологической проблемой.

5.4. Будущие направления. Функции влияния для LLM — это активно развивающаяся область исследований с множеством перспективных направлений.

Атрибуция для мультимодальных моделей. Современные базовые модели всё чаще являются мультимодальными, обучаясь на комбинациях текста, изображений, аудио и других модальностей. Адаптация функций влияния к мультимодальному контексту требует решения новых задач: как атрибутировать влияние пар изображение-текст? как учитывать взаимодействия между модальностями? как эффективно вычислять градиенты для больших изображений или видео? Эти вопросы открывают новые исследовательские направления.

Атрибуция для моделей с retrieval-augmented generation (RAG). Многие современные приложения LLM используют RAG-архитектуры, которые дополняют параметрические знания модели

информацией, извлечённой из внешних баз данных во время инференса. Для таких систем естественно возникает вопрос атрибуции: какая часть ответа модели определяется её обучающими данными (параметрическими знаниями), а какая — извлечёнными документами? Разработка методов для декомпозиции и атрибуции влияния в RAG-системах является важной практической задачей.

Дифференциальная приватность и атрибуция. По мере роста озабоченности вопросами приватности растёт интерес к обучению моделей с гарантиями дифференциальной приватности. Однако механизмы, обеспечивающие дифференциальную приватность (например, добавление шума к градиентам), взаимодействуют с функциями влияния нетривиальным образом: добавленный шум меняет поверхность потерь и гессиан, что влияет на оценки влияния. Понимание того, как функции влияния ведут себя в режимах с дифференциальной приватностью, и разработка методов атрибуции, совместимых с приватностью, является важным направлением, особенно для применений в чувствительных доменах.

Активное обучение и оптимальный отбор данных. Функции влияния предоставляют оценки полезности данных, которые можно использовать для оптимизации состава обучающего набора. В контексте активного обучения или curriculum learning можно использовать предсказания функций влияния для выбора, какие данные показать модели на следующем этапе обучения. Разработка теоретически обоснованных стратегий активного отбора данных на основе функций влияния, с гарантиями сходимости и эффективности обучения, является перспективным направлением исследований.

Интеграция с другими методами интерпретируемости. Функции влияния — это лишь один из многих инструментов интерпретируемости машинного обучения. Другие методы — например, карты внимания, методы атрибуции признаков (saliency maps), анализ представлений — предоставляют комплементарные взгляды на работу моделей. Интеграция функций влияния с этими методами могла бы дать более целостное понимание. Например, можно использовать функции влияния для идентификации влиятельных обучающих примеров, а затем анализировать карты внимания для понимания, какие конкретные части этих примеров модель “запомнила”. Разработка унифицированных фреймворков интерпретируемости, комбинирующих различные методы, является важной задачей.

Применения к безопасности и аудиту. По мере того как LLM становятся критически важными компонентами реальных систем, растут требования к их безопасности, надёжности и соответствию регуляциям. Функции влияния могут играть ключевую роль в:

- обнаружении и удалении защищённых авторским правом данных из обучающих корпусов;
- аудите моделей на предмет использования приватной или чувствительной информации;
- идентификации и устранении вредоносных данных (backdoor attacks, data poisoning);
- объяснении решений модели в регулируемых областях (медицина, финансы, право);
- верификации соответствия моделей этическим принципам и регуляторным требованиям.

Развитие методов функций влияния специально для этих применений, с учётом их уникальных требований к точности, надёжности и интерпретируемости, также представляется важным направлением.

5.5. Заключительные замечания. Функции влияния представляют собой мощный инструмент для понимания связи между обучающими данными и поведением обученных моделей. В эпоху больших языковых моделей, когда огромные корпуса данных преобразуются в сложные системы с триллионами параметров, способность атрибутировать выходы моделей к конкретным обучающим данным становится не просто научным любопытством, но практической необходимостью, будь то для отладки неожиданного поведения, для соответствия юридическим требованиям, для улучшения качества обучающих данных или для понимания фундаментальных механизмов обучения.

Развитие масштабируемых аппроксимаций сделало функции влияния практически возможными даже для крупнейших современных моделей. В сочетании с методами фильтрации, рассмотренными в первой части обзора, они образуют полный арсенал инструментов для работы с обучающими данными для больших языковых моделей.

Мы надеемся, что настоящий обзор послужит полезным введением в эту область как для исследователей, стремящихся развивать новые методы, так и для практиков, выбирающих инструменты для конкретных задач. По мере того как большие языковые модели продолжают

трансформировать искусственный интеллект и общество в целом, понимание связи между данными и поведением моделей будет становиться всё более важным. Функции влияния предоставляют один из ключевых инструментов для этого понимания.

Список литературы

1. N. Agarwal, B. Bullins, and E. Hazan, *Second-Order Stochastic Optimization for Machine Learning in Linear Time*, Journal of Machine Learning Research **18** (2017), no. 116, 1–40.
2. S. K. Choe, H. Ahn, J. Bae, K. Zhao, M. Kang, Y. Chung, A. Pratapa, W. Neiswanger, E. Strubell, T. Mitamura, J. Schneider, E. Hovy, R. Grosse, and E. Xing, *What Is Your Data Worth to GPT? LLM-Scale Data Valuation with Influence Functions*, 2024.
3. R. Cook, D. Weisberg, *Residuals and Influence in Regression*, Chapman and Hall, 1982.
4. T. George, C. Laurent, X. Bouthillier, N. Ballas, and P. Vincent, *Fast Approximate Natural Gradient Descent in a Kronecker-Factored Eigenbasis*, 2021.
5. R. Grosse, *et al.*, *Studying Large Language Model Generalization with Influence Functions*, 2023.
6. H. Guo, N. Rajani, P. Hase, M. Bansal, and C. Xiong, *FastIF: Scalable Influence Functions for Efficient Model Interpretation and Debugging*, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Association for Computational Linguistics, Nov. 2021, pp. 10333–10350.
7. F. R. Hampel, *The Influence Curve and Its Role in Robust Estimation*, Journal of the American Statistical Association **69** (1974), no. 346, 383–393.
8. P. W. Koh and P. Liang, *Understanding Black-Box Predictions via Influence Functions*, in: Proceedings of the 34th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 70, PMLR, 06–11 Aug 2017, pp. 1885–1894.
9. H. Lin, J. Long, Z. Xu, and W. Zhao, *Token-Wise Influential Training Data Retrieval for Large Language Models*, in: Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, 2024, pp. 841–860.
10. J. Martens and R. Grosse, *Optimizing Neural Networks with Kronecker-Factored Approximate Curvature*, in: Proceedings of the 32nd International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 37, PMLR, 07–09 Jul 2015, pp. 2408–2417.
11. Y. Pan, H. Lin, Y. Ran, J. Chen, X. Yu, W. Zhao, D. Zhang, and Z. Xu, *ALinFiK: Learning to Approximate Linearized Future Influence Kernel for Scalable Third-Parity LLM Data Valuation*, in: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), Association for Computational Linguistics, Apr. 2025, pp. 11756–11771.

12. S. M. Park, K. Georgiev, A. Ilyas, G. Leclerc, and A. Madry, *TRAK: Attributing Model Behavior at Scale*, in: Proceedings of the 40th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol. 202, PMLR, 23–29 Jul 2023, pp. 27074–27113.
13. A. Schioppa, P. Zablotskaia, D. Vilar, and A. Sokolov, *Scaling Up Influence Functions*, in: Proceedings of the AAAI Conference on Artificial Intelligence **36** (2022), no. 8, 8179–8186.

Nikolenko S. I. Influence functions for data evaluation in large language models.

Influence functions are a mathematical tool for estimating how individual training examples affect the behavior of trained machine learning models. Unlike the data filtering and quality evaluation methods reviewed in the first part of this survey, influence functions provide a principled approximation of counterfactual leave-one-out estimates through local linearization and accounting for the geometry of the parameter space. This survey systematically presents the theory of influence functions: from the classical derivation via upweighting and the implicit function theorem to modern scalable approximations. We provide a detailed analysis of methods for computing inverse Hessian-vector products (LiSSA, K-FAC/EK-FAC) and present techniques that make data attribution practically feasible for models with billions of parameters: projection-based methods (TRAK, LoGra), first-order methods (RapidIn), and early-stage training approaches (LinFIK, ALinFIK). We conclude by discussing the limitations of influence functions, open problems, and promising research directions, including applications to multimodal models, RAG architectures, and safety auditing tasks.

Санкт-Петербургское отделение
Математического института
им. В. А. Стеклова РАН,
наб. р. Фонтанки, 27,
191023, Санкт-Петербург, Россия
E-mail: sergey@logic.pdmi.ras.ru

Поступило 10 января 2026 г.