universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Select or Project?
Evaluating Lower-dimensional Vectors for
LLM Training Data Explanations“

verfasst von / submitted by

Ing. Lukas Hinterleitner BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2025 / Vienna, 2025

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von / Supervisor:	Univ.-Prof. Dr. -Ing. Benjamin Roth, B.Sc. M.Sc.

Acknowledgements

First and foremost, I want to express my gratitude to Ben, for assisting me during working on my master thesis and for taking the time to discuss present issues. Also a big thanks to Loris, a PHD student under Ben's supervision, for providing answers for my questions. Furthermore, I am very grateful to my family for supporting me during my complete studies, either financially or emotionally. Another thanks to Huawei Lin for providing me the illustration templates for their paper on Token-wise Influential Training Data Retrieval for Large Language Models. And last but not least, a big sarcastic thanks to my friends for asking me a lot why I have not yet finished my master thesis that took a little bit longer than expected. Jokes aside, I am very grateful to have such a large support network of family and friends.

Abstract

Large Language Models (LLMs) are largely opaque, making it difficult to identify which training examples influence a given prediction. Instance-based explanation methods that rely on model gradients are computationally challenging due to the high dimensionality of modern LLMs. This thesis investigates whether lower-dimensional gradient surrogates, constructed either by **selecting** a subset of layer components or by **projecting** the full gradient, can effectively reproduce instance-level explanations. The central question is: Is it more effective to select or to project?

A novel and efficient retrieval-based benchmark is proposed to evaluate instance-based explanations. The task is to identify an original training sample from its paraphrased version by measuring the cosine similarity of their respective loss gradients. This benchmark is used to compare two strategies for creating low-dimensional gradient representations on a 1.2B parameter LLM: (1) a greedy forward selection algorithm that identifies a small, architecturally-informed subset of influential layer components, and (2) random projection, which maps the full gradient to a dense, lower-dimensional space.

The results demonstrate that, for the retrieval task, a targeted selection of a small subset of gradient components is superior. A greedily selected subset, often comprising less than 5% of the model's parameters and drawing heavily from MLP and attention query/output blocks, not only outperforms random projections, but also achieves higher retrieval accuracy than the full model gradient itself. This suggests that the full gradient can be a noisy and suboptimal signal for this task. Although random projection effectively preserves the global geometry of the full-gradient space, it is computationally more expensive and less accurate for the specific retrieval objective.

The findings provide a clear answer: for building efficient and effective instance-based explanation tools framed as a retrieval task, selecting a small, architecturally-aware representation of the gradient is more accurate and computationally efficient than projecting the entire gradient. This work contributes an efficient benchmark for instance-based explanations, a principled comparison of dimensionality reduction strategies, and empirical evidence advocating for targeted selection over holistic projection.

Kurzfassung

Große Sprachmodelle (LLMs) sind weitgehend intransparent, was die Identifizierung jener Trainingsbeispiele erschwert, die eine bestimmte Vorhersage beeinflussen. Instanzbasierte Erklärungsmethoden, die auf Modellgradienten basieren, stellen aufgrund der hohen Dimensionalität moderner LLMs eine große rechnerische Herausforderung dar. Diese Arbeit untersucht, ob niedrigdimensionale Gradienten-Surrogate, die entweder durch die **Auswahl** (Selektion) einer Teilmenge von Schichtkomponenten oder durch die **Projektion** des gesamten Gradienten erstellt werden, instanzbasierte Erklärungen effektiv reproduzieren können. Die zentrale Fragestellung lautet: Ist es effektiver, auszuwählen oder zu projizieren?

Zur Evaluierung instanzbasierter Erklärungen wird ein neuartiges, effizientes retrievalbasiertes Benchmark vorgestellt. Die Aufgabe besteht darin, ein originales Trainingsbeispiel anhand seiner paraphrasierten Version zu identifizieren, indem die Kosinus-Ähnlichkeit der jeweiligen Kostengradienten gemessen wird. Dieses Benchmark wird verwendet, um zwei Strategien zur Erstellung niedrigdimensionaler Gradientenrepräsentationen auf einem 1,2B-Parameter-LLM zu vergleichen: (1) ein gieriger Vorwärtsauswahl-Algorithmus, der eine kleine, architekturbasiert informierte Teilmenge einflussreicher Layerkomponenten identifiziert, und (2) die zufällige Projektion, die den vollständigen Gradienten auf einen dichten, niedrigdimensionalen Raum abbildet.

Die Ergebnisse zeigen, dass für die Retrieval-Aufgabe eine gezielte Auswahl einer kleinen Teilmenge von Gradientenkomponenten überlegen ist. Eine gierig ausgewählte Teilmenge, die oft weniger als 5 % der Modellparameter umfasst und hauptsächlich aus MLP- und Attention-Query/Output-Blöcken besteht, übertrifft nicht nur zufällige Projektionen, sondern erzielt auch eine höhere Abrufgenauigkeit als der vollständige Modellgradient selbst. Dies deutet darauf hin, dass der vollständige Gradient für diese Aufgabe ein verrauschtes und suboptimales Signal sein kann. Während die zufällige Projektion die globale Geometrie des Gradientenraums effektiv bewahrt, ist sie für das spezifische Abrufziel rechenintensiver und ungenauer.

Die Resultate liefern eine klare Antwort: Für die Entwicklung effizienter und effektiver instanzbasierter Erklärungswerkzeuge, die als Retrieval-Aufgabe konzipiert sind, ist die Auswahl einer kleinen, architekturbewussten Repräsentation des Gradienten genauer und recheneffizienter als die Projektion des gesamten Gradienten. Diese Arbeit leistet einen Beitrag durch ein effizientes Benchmark für instanzbasierte Erklärungen, einen prinzipiengeleiteten Vergleich von Dimensionsreduktionsstrategien und empirische Belege, die eine gezielte Auswahl gegenüber einer holistischen Projektion befürworten.

Contents

List of Tables

List of Figures

List of Algorithms

Listings

1. Introduction

Large language models (LLMs) are now embedded in everyday software. Yet their training and inference pipelines remain largely opaque, raising the question: *Which specific training examples matter for a given prediction, and can they be identified efficiently?* This thesis evaluates whether lower-dimensional gradient surrogates, constructed either by *selecting* a subset of layer components or by *projecting* full gradients, can reproduce instance-level, gradient-based explanations as accurately as the full model gradient. In short: Select or project?

In recent years, machine learning applications have gained increased popularity, especially with the emergence of Large Language Models (LLMs) such as BERT [DCLT19], GPT [BMR+20, Ope23], and LLaMA [TMS+23, DJP+24], etc. These LLMs employ deep neural networks (DNNs), Transformers [VSP+17], and other components to generate coherent text based on a given textual (and occasionally visual) input. That being said, the subsequent sections will focus strictly on text-to-text language models, specifically auto-regressive language models such as GPT-1 proposed in [RNSS18].

1.1. Black Boxes

Unlike traditional machine learning models, such as Decision Trees or Linear Regression, DNNs and thus LLMs are so-called black boxes. For these black boxes, it is not straightforward to determine how or why such models arrive at a particular conclusion, prediction, or classification. This lack of transparency poses significant challenges in critical fields, for example healthcare, where understanding the reason behind each decision is highly important due to the responsibility towards patients [XLZ+24]. Moreover, EU's General Data Protection Regulation (GDPR) explicitly grants individuals the right to obtain meaningful information regarding the logic that underlies automated decisions in information systems [Eur16]. Explanations are therefore important not just for end users affected by these decisions but also for engineers and researchers who develop, test, and refine such systems.

As a direct consequence, the explainability for Artificial Intelligence (AI) has become a critical research topic, gaining more and more attention under the name Explainable Artificial Intelligence (XAI) [CCWB21].

1.2. Explainable Artificial Intelligence

As noted above, the aim of XAI is to develop models that can be interpreted by humans, particularly for sensitive applications in fields such as military, finance, and healthcare. Therefore, it is a key area of research for creating transparent and traceable AI systems.

Before proceeding, we must distinguish between **explainability** and **interpretability**.

- **Interpretability:** Interpretability enables developers to get a better understanding of how a specific AI-model works.
- **Explainability:** Explainability provides insight into how the system came up with a certain decision that is understandable to the end user. Explainable models provide explanations in a way that users understand why the model generated a particular conclusion.

In this work, the focus is predominantly on **explainability** rather than on **interpretability**, although these domains frequently overlap [AAE+23]. Since there are multiple approaches to XAI, a quick overview helps to categorize these approaches. According to [ZCY+24] LLM explainability can be structured as presented in Figure 1.1. The focus for this thesis is on the *traditional fine-tuning paradigm*, more on that later. In this context, the key distinction between *local explanation* and *global explanation* is that the former aims to clarify individual model predictions, while the latter seeks to shed light on the internal workings of language models.

The method introduced in the upcoming chapters falls under the *local explanation* category and specifically under *example-based explanation*, as it targets individual training instances rather than isolated features [ZCY+24].

1.3. Problem Statement

As mentioned above, this work focuses on example-based (instance-based) explanations. However, while instance-based explanations are well known in the context of classification models, they have been relatively unexplored for Large Language Models. Given that Large Language Models can contain many parameters (for example, LLaMA 3.1 has up to 405 billion parameters [AI24]), the associated computations can be very expensive. Specifically, tracing a prediction back to a training instance often requires calculating at least one gradient for each layer per training sample.

1.3.1. Methodological Overview: A Retrieval-Based Benchmark

To systematically evaluate instance-based explanations under these constraints, this thesis introduces an efficient, **retrieval-based benchmark**. The core methodology is designed to answer a fundamental question: Can a low-dimensional gradient surrogate effectively

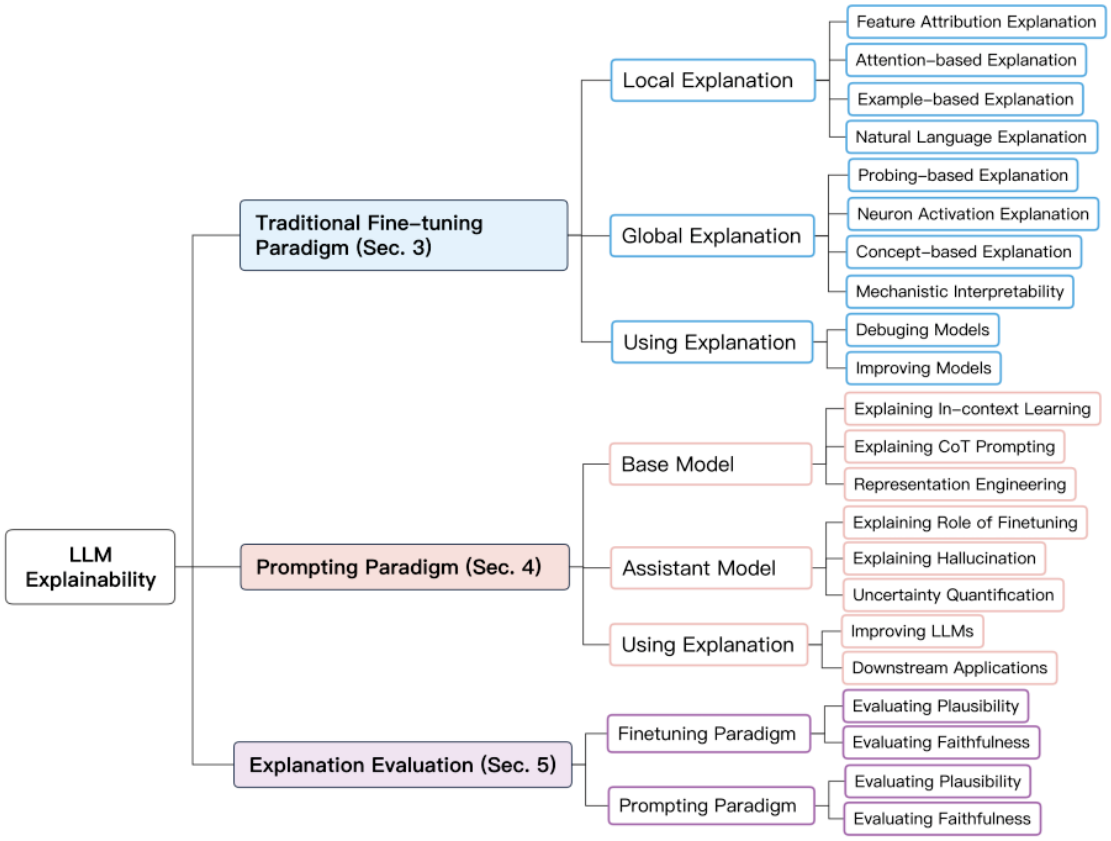


Figure 1.1.: Categories of LLM explainability adopted from [ZCY+24].

identify the original training data point? This identification is evaluated when the model is queried with an input that is semantically equivalent to the original training data point but lexically distinct. This approach assesses the robustness and efficiency of gradient-based explanations under realistic conditions where exact input matches are unlikely.

To create these queries, each sample in a supervised fine-tuning dataset is **paraphrased** using an external model, creating a parallel dataset of (original, paraphrase) pairs. This process, which forms the basis of the benchmark's evaluation datasets, is detailed in section 3.3. This paraphrasing strategy is chosen for two primary reasons. First, it directly tests whether a gradient-based explanation is robust to superficial lexical changes while capturing underlying semantic meaning, a key requirement for any useful explanation. Second, it provides an evaluation setup that is **re-training free**. Unlike other evaluation strategies that require "expensive additional fine-tuning steps" [ABL+22] that are "infeasible to perform with LLMs", this approach only computes how the model's loss gradient changes in response to paraphrased inputs. This makes it particularly well-suited for evaluating large, already fine-tuned LLMs. The underlying assumption is that a

semantically faithful paraphrase should elicit a similar "influence" from the model as its original counterpart.

The "influence" itself is measured using **gradient cosine similarity**. The original and paraphrased samples are passed through the final, trained model, and their respective gradients are calculated. The cosine similarity is then employed to measure the degree to which these gradients align. As formalized in section 3.5, a high similarity score is interpreted as a successful "retrieval" of the original instance.

1.3.2. The Core Challenge: High-Dimensional Gradients

The primary obstacle to this approach is the immense size of the full model gradient. For a single sample, the combined gradients can be as large as the entire model. In the case of LLaMA 3.1 (405B), the gradients of a single example contain up to 405 billion floating-point values, which adds up to $\approx 1.5\,\text{TB}$ of storage requirements per sample.

Comparing the gradients for every paraphrased sample against every original sample, an $\mathcal{O}(N^2)$ operation, is therefore computationally infeasible. To make this comparison tractable, the search space must be drastically reduced. This is achieved by using the **BM25** ranking function [RZ09]. As detailed in subsection 3.6.1, BM25 is used as an efficient, text-based heuristic to create a small, fixed-size set of "distractor" candidates for each query. This reduces the number of expensive gradient comparisons from $N \times N$ to $N \times b$, where $b \ll N$. The motivation for this step is purely computational: it allows the benchmark to remain feasible by focusing the expensive gradient-similarity calculations only on a small set of lexically plausible candidates.

1.3.3. Research Question: Select or Project?

Even with a reduced candidate set, working with full gradients is prohibitive. A common strategy to manage this complexity is to use a low-dimensional **gradient surrogate**. Previous work has often made different choices, such as focusing exclusively on the final layer or the MLP blocks [SZVS22, ABL$^+$22, GBA$^+$23], under the assumption that these components are most critical. While computationally convenient, such pre-selections may discard crucial information.

This thesis moves beyond that by systematically evaluating two competing strategies for creating these surrogates, directly addressing the central research question: *Select or project?*

- **Greedy Layer Selection (Select):** Instead of using the entire gradient, can we identify a small subset of important layer components that are most influential? This architecture-aware approach, detailed in section 3.7, builds a sparse representation by greedily selecting layers whose gradients are most informative for the retrieval task.

- **Random Projection (Project):** As a baseline, can a dense, low-dimensional representation of the full gradient preserve enough information? This method, formally introduced in subsection 3.7.5, uses random projections to map the entire gradient vector into a smaller space while approximately preserving its geometry.

By systematically comparing these two strategies, this thesis provides a direction to determine whether it is more effective to strategically **select** a few highly influential components or to **project** the information from all components into a compact representation. An overview of the proposed approach is illustrated in Figure 1.2.

The main contributions of this work are threefold: it (1) proposes an efficient retrieval-based benchmark for instance-based explanations; (2) conducts a principled comparison of architecture-aware selection versus geometry-preserving projection for gradient approximation; and (3) demonstrates empirically on a 1.2B parameter LLM that targeted selection is more accurate and more computationally efficient. The central insight is that for instance-based explanation, a small, architecturally-informed representation of the gradient provides a clearer and more discriminative signal than the full gradient itself.

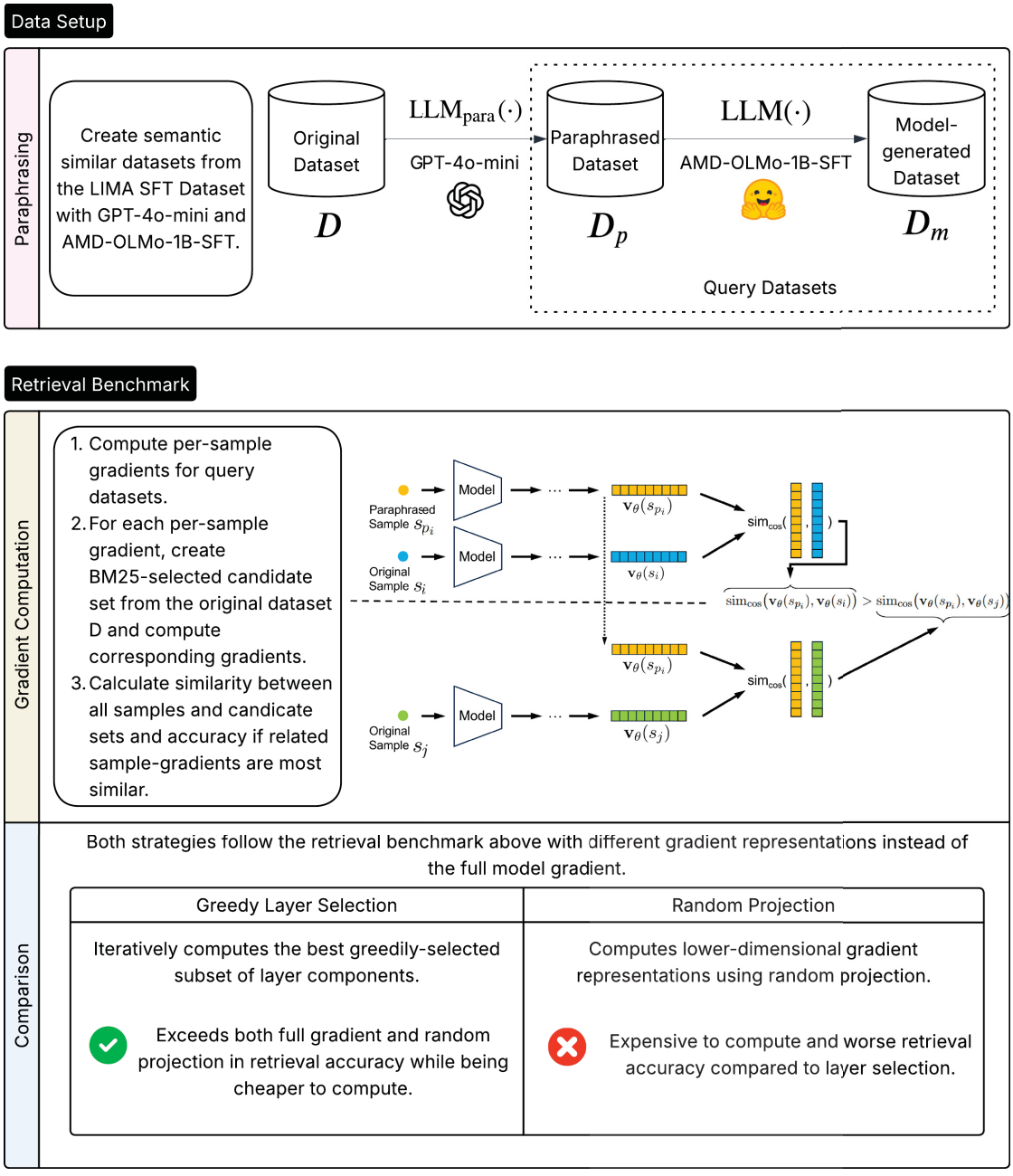


Figure 1.2.: Overview of the proposed approach in this thesis.

2. Related Work

According to [ZCY+24], the training of LLMs can be categorized into two paradigms, namely *traditional fine-tuning* and *prompting* depending on how they are adapted for downstream tasks (see Figure 1.1).

In *traditional fine-tuning*, a Large Language Model is pre-trained on a large corpus of unlabeled text data and further fine-tuned on labeled data tailored to a specific domain and benchmarked accordingly (for example, the GLUE benchmark [WSM+19]). In the *prompting* paradigm the so-called *prompts* enable the model to behave in a certain way without retraining it, enabling zero-shot, one-shot and few-shot learning without requiring additional training data [BMR+20].

As stated in section 1.2 and shown in Figure 1.1, *Gradient-Based Explanation* is a subcategory of *feature attribution-based explanation*. However, the approach presented in this work falls under *example-based explanation* (or in other words *instance-based explanation*) but also uses a kind of gradient-based solution and therefore *gradient-based explanation* will be briefly described in the next section.

2.1. Gradient-Based Explanation

The importance of each input feature can be evaluated using gradient-based attribution techniques. This is achieved by analyzing the partial derivatives of the output with respect to each input dimension. The magnitude of the derivatives reveals how sensitive the output is to changes in the input. This method can generally be formulated as $s_j = \frac{\partial f(x)}{\partial x_j}$, where $f(x)$ is the function of the neural network and x_j is the input vector [ZCY+24].

However, the approach covered in this work will use the full gradients of each model layer with respect to a single training example instead of a single feature dimension; hence, it is more important to shed more light on example-based explanations and, therefore, different example-based variants will be briefly discussed in more detail below. Not all of them are actually completely related to the proposed approach in this work; however, it still makes sense to establish an outline of the closely and not so closely related topics to gain an overview before jumping into details of the proposed approach.

Nevertheless, not all gradient-based explanation methods are tied to different feature dimensions, and gradients can also be useful in the context of example-based explanation. More about that is outlined in subsection 2.2.2.

2.2. Example-Based Explanation

Example-based explanations offer a way to explain a model's behavior based on individual examples (or instances) [KL17]. In other words, they offer a way to identify how a model's output changes on the basis of different inputs. Furthermore, according to [ZCY+24], example-based explanations can be divided into three categories: *adversarial examples*, *counterfactual explanations*, and *data influence*.

2.2.1. Adversarial Example

Language Models have been shown to be vulnerable to small adjustments in their input data. Although these carefully constructed changes may not be noticeable to humans, they can influence the output of a model. In the beginning, adversarial examples were based on word-level alterations such as small errors, removal, and insertion. Over time, more advanced token-level perturbation methods such as *TextFooler* [JJZS20] were published. These approaches specify important words on embedding similarity, same part of speech, etc. and are correspondingly modified. However, methods relying on word embeddings are limited in sentence representation compared to contextualized representations, which often lead to incoherent pieces. [WXL+22] proposed *SemAttack*, a framework that can be used for different embedding spaces, such as type space, knowledge space, and contextualized semantic space. Once the input tokens are transformed to an embedding space, the perturbed embeddings are optimized iteratively until a specific goal is met. One of the experiments shows that by replacing 5% of the words, the accuracy of BERT drops from 70.6% to 2.4%.

2.2.2. Data Influence

The notion of *data influence* originally came from statistics. There, the influence of a particular data point on a model's parameters was measured by simply removing it during training. This concept, often called *leave-one-out influence*, is a dominant framework for instance-based explanation. However, since retraining a model for every single data point is computationally infeasible, this effect is often approximated with re-training free methods. Regarding data influence methods, there are two important distinctions based on when gradients are obtained:

Static, Gradient-Based Influence Estimation

This type of explanation method got its name because it measures the influence only on the final model parameters θ_T, which are static after training is finished. Compared to dynamic methods (see Dynamic, Gradient-Based Influence Estimation below), the static estimator is computationally more efficient since it only relies on the final model parameters. However, it has less information to estimate and therefore provides limited insight. The approach proposed in this work also falls into this category, as it depends on an already trained or fine-tuned model and calculates gradients with respect to single

input examples [HL24].

Using **influence functions**, a static data influence estimator, the authors of [KL17] are able to explain the predictions of a black-box model by tracing back to the specific training examples that contributed the most to each prediction. To make this setting suitable for large-scale deep learning models, their approach relies on approximating these influence functions by using only gradient and Hessian-vector products.

The contributors of [GRH+21] demonstrate an improved application of influence functions for model debugging by using k-Nearest Neighbors (k-NN) to narrow down the search space to a subset of good candidate data points. However, computing a high-dimensional inverse Hessian-vector product is very expensive and, therefore, influence functions are difficult to scale to Large Language Models. Recent applications therefore restrict computation to a subset of model parameters, such as the multilayer perceptron (MLP) layers [GBA+23] or Low-rank adaptation (LoRA) layers [KWWZ23].

Consequently, in 2023, Anthropic scaled the idea of influence functions for LLMs with up to 52 billion parameters by an Eigenvalue-corrected Kronecker-Factored Approximate Curvature (EK-FAC) approximation. In their work, they use influence functions to examine generalization patterns of Large Language Models and evaluate their approximation by reporting agreement with the *proximal Bergmann response function*. According to the authors, influences approach near zero when the order of the key phrases in a prompt is changed [GBA+23]. Other evaluation strategies require expensive additional fine-tuning steps. For example, [ABL+22] evaluate by introducing facts with the fine-tuning data that the model did not know before and then verify whether influence estimation methods correctly attribute importance to this fine-tuning set when queried about them. Such methods are often infeasible to perform with Large Language Models.

Dynamic, Gradient-Based Influence Estimation

Dynamic gradient-based estimators approach the analysis by reconstructing the training's data influence based on the model parameters during training, e.g. $\theta_0, ..., \theta_T$. An example of such an approach is TracInCP [PLKS20], which aims to approximate training data influence by comparing a test instance's loss gradient to the gradients of training instances at a set of model checkpoints. In the case of Large Language Models, working with the model parameters for each training checkpoint is more computationally expensive. However, they can be more precise than static estimators because they can harness more underlying information about the changes in the model with respect to the training data [HL24].

2.3. Paraphrased Corpora Analysis

The authors of [WMSJ25] did an experiment that is similar to the approach proposed in this work, providing a re-training free evaluation setup. There, the robustness of In-Run Data Shapley and influence functions is evaluated by identifying relevant individual corpora that have been paraphrased. They also calculate a rank based on the original training corpus and its paraphrased versions. For comparison, the BM25 [RZ09] distance is used as an oracle to assess the lexical similarity between the paraphrased versions and the original corpora, rather than the semantic similarity. Even with completely rewritten corpora, the corresponding original samples still rank very high according to In-Run Data Shapley and influence functions compared to BM25. This method allows for a systematic comparison of different strategies for obtaining instance-based explanations without costly re-training.

2.4. Dimensionality and Performance

The vast number of parameters in a modern LLM, often in the billions, makes analyzing the full gradient computationally prohibitive and potentially noisy. Therefore, selecting a subset of parameters is a common and necessary strategy to make instance-based gradient explanations feasible. This selection can be approached through traditional dimensionality reduction techniques or, more strategically, by focusing on specific, functionally significant components of the model's architecture, such as individual layers.

2.4.1. Traditional Dimensionality Reduction Techniques

One approach to manage the high dimensionality of gradients is to use methods like *random projection* [JL84]. This technique projects the high-dimensional gradient vector into a lower-dimensional space while approximately preserving the distances between vectors. While computationally efficient, random projection is data-agnostic and may obscure the fine-grained information encoded in the original gradient, as it does not consider the underlying structure of the neural network.

2.4.2. Layer Selection

A more informed approach to dimensionality reduction leverages the inherent structure of the model itself. Modern LLMs are predominantly built upon the Transformer architecture [VSP+17], which is composed of a stack of identical layers. Each layer, or block, transforms its input representations through a multi-head self-attention mechanism and a position-wise feed-forward network. This deep, layered structure is not monolithic; instead, a functional specialization emerges across the network's depth.

Research has shown that different layers capture different levels of linguistic abstraction, progressing from surface-level features to complex semantic representations [TDP19, JSS19]. Generally, the *initial layers* (closer to the input) tend to learn surface properties

like word order and basic syntax. The *middle layers* capture more complex syntactic and local semantic relationships, while the *final layers* (closer to the output) synthesize this information into highly contextualized, task-specific representations.

This hierarchical processing has significant implications for gradient-based explanation methods. The choice of which layer's parameters to use for gradient computation is not trivial, as gradients with respect to different layers can reveal which level of abstraction, syntactic, semantic, or otherwise, is most critical for a model's specific prediction. This raises a key question for instance-based explainability: which layer provides the most stable, informative, and computationally efficient signal for understanding model behavior?

Are Early Layers Better?

The common practice of restricting influence analysis to the final layer [SZVS22, ABL+22] or just MLP layers [GBA+23] of a network, often chosen for computational convenience, has been challenged by recent work. The authors of [YTS+22] argue that for LLMs, the final layer is a suboptimal choice for calculating data influence due to a phenomenon they call *cancellation effect*. This effect occurs when influence scores derived from later layers exhibit large magnitudes but conflict with one another, thereby reducing the discriminative power of the explanation method. The authors attribute this to the abstract, high-level features encoded in the network's deeper layers. In these layers, distinct inputs often activate the same neurons or pathways corresponding to generalized concepts, causing their gradients to become more aligned and, therefore, more susceptible to cancellation.

Consequently, they propose that influence should instead be computed on the parameters of the first layer, the word embedding layer in most LLMs. Early layers are presumed to encode features that are more faithful to the specific tokens and structure of an individual input, rather than the highly abstract concepts found in deeper layers. This specificity makes them less susceptible to the cancellation effect. While the word embedding layer itself may not capture high-level semantic information, its *gradients* do. Because gradients are back-propagated through the entire network via the chain rule, the gradients with respect to the embedding weights implicitly incorporate information from all subsequent layers.

Empirical evaluations in [YTS+22] support this "first is better than last" hypothesis. Using a case deletion metric, they demonstrate that an influence method operating on the word embedding layer significantly outperforms equivalent methods applied to the final layer. Their analysis reveals a clear trend: the quality of the influence score degrades as it is computed on progressively later layers of the model, with the first layer providing the most reliable explanations. The authors provide a strong, empirically grounded rationale for selecting a specific architectural component, the embedding layer, as a cornerstone for constructing efficient and effective instance-based explanations. With this in mind, the following thesis will also shed light onto different layer depths and their effect on instance-

and gradient-based explanations.

3. Gradient Similarity as a Method for Explanation

As outlined in the methodological overview in section 1.3, the core problem of this thesis is to find an efficient, low-dimensional surrogate for full-model gradients that can reliably identify influential training instances. This chapter now formally introduces the core methodology that forms the basis of the retrieval-based benchmark used to address this problem. It details the components of the benchmark, including the formal fine-tuning setup (section 3.2), the construction of the paraphrased and model-generated datasets (section 3.3), and the mathematical definition of gradient similarity used for retrieval (section 3.5). Finally, it presents the technical implementation of the two competing strategies for dimensionality reduction that are central to this work: greedy layer selection (section 3.7) and random projection (subsection 3.7.5).

3.1. Instance-based Explainability Methods

To explain model predictions, this work adopts an instance-based view, which traces a model's behavior back to the training examples that influenced it most [KL17]. We define an example's "influence" as the gradient of the per-sample loss, computed on the final, frozen model. Our method then compares a query, a paraphrased version of a training sample, to the original data by calculating the cosine similarity of their flattened gradient vectors. This metric serves as an affinity measure, quantifying the directional alignment of the parameter updates that each sample would induce. The central assumption is that a paraphrase and its original will display highly aligned gradients, making this similarity a direct signal for identifying the source instance (visualized in Figure 3.2 and formalized in Equation 3.6).

This framing is (i) *local*, targeting decisions at the level of single instances, (ii) *static*, using the final checkpoint without reconstructing training dynamics (subsubsection 2.2.2), and (iii) *model-agnostic*, applicable to any architecture that provides per-sample gradients of a language-modeling loss. It also *does not require retraining*: only the change in the loss gradient under paraphrasing is evaluated. Explanation quality is assessed via a retrieval task: for each paraphrase, identify its original among BM25-selected candidates (subsection 3.6.1) and report accuracy as in Equation 3.7–Equation 3.8. The remainder of the chapter formalizes the fine-tuning setup (section 3.2), the construction of paraphrased/model-generated datasets (section 3.3), the gradient representation (subsection 3.5.1). Last but not least, low-dimensional representations via a greedy layer selection

and random projections (section 3.7–subsection 3.7.5) are evaluated in this setup.

3.2. Fine-Tuning Setting

Given a fine-tuning training dataset $D = \{s_i\}_{i=1}^{N}$, where s_i is the i-th sample that contains a sequence of tokens. In the scope of this work, the term *original* refers to samples in this dataset. Each of these samples contains instructions and generations, where $s_i^{-P_i}, ..., s_i^0$ denote the tokens of the i-th instruction prompt of length P_i, and $s_i^1, ..., s_i^{G_i}$ denote the tokens of the i-th generation for the i-th prompt with a length of G_i.

Then the training of an LLM can be formulated as follows:

$$\hat{\theta} = \arg\min_{\theta} \frac{1}{\sum_{i=1}^{N} G_i} \sum_{i=1}^{N} \sum_{j=1}^{G_i} \mathcal{L}(s_i^j, \theta) \tag{3.1}$$

where θ represents the parameters of the model and $\mathcal{L}(\cdot, \theta)$ is the loss function. The term $\frac{1}{\sum_{i=1}^{N} G_i}$ in Equation 3.1 is the total number of generation tokens in the entire training set. Usually, for the LLM objective, the loss is $\mathcal{L}(s_i^j, \theta) = -\log p_\theta(s_i^j \mid s_i^{-P_i}, \ldots, s_i^{j-1})$ the negative log-likelihood of the correct token given all previous tokens. Modern LLM architectures use adapted versions of it or may use other loss functions. However, in this setting, the choice of loss function does not matter as long as it suits language modeling. Let $\mathcal{L}(s_i, \theta) = \frac{1}{G_i} \sum_{j=1}^{G_i} \mathcal{L}(s_i^j, \theta)$, be the loss per sample which will be relevant in the following sections.

Usually, machine learning models are trained using batches of training data. In the setting of this work, the batch size is set to exactly one to measure phenomena for single instances. Hence, in the training process, for iteration a, the parameters are updated from θ_{a-1} to θ_a depending only on the training sample s_a.

Modern LLM training still relies on the original idea of gradient descent to update the model weights despite being more advanced in terms of convergence, regularization, memory-efficiency, etc., for example, AdamW [LH19] or LAMB [YLR+20]. However, the proposed approach only considers the gradients of the loss function with respect to each individual training sample, without updating the model weights correspondingly; therefore, establishing the idea of gradient descent is sufficient here. The general formulation of gradient descent is shown in Equation 3.2:

$$\theta_{a+1} = \theta_a - \eta_a \nabla_\theta \mathcal{L}(s_a, \theta_a) \tag{3.2}$$

where η_a is the learning rate in iteration a and $\nabla_\theta \mathcal{L}(s_a, \theta_a)$ denotes the gradient of the loss function given s_a and the current model parameters θ_a with respect to θ [LLXZ24]. In other words, the gradients determine how the model weights are updated based on the given training data. Hence, the gradient $\nabla_\theta \mathcal{L}(s_a, \theta_a)$ contains information on how the

model parameters θ will be updated depending on the training data sample s_a. With this in mind, the effect of the gradient based on single samples will be analyzed in detail in the following chapters.

3.3. Paraphrasing a Dataset

As introduced in the methodological overview (subsection 1.3.1), the core of the evaluation benchmark relies on paraphrasing. This section formally describes this process. The idea is to take a dataset D on which a model was fine-tuned and paraphrase it using another LLM. When paraphrasing, we obtain $D_p = \{s_{p_i}\}_{i=1}^{N}$, a dataset where the inputs and the outputs, in other words, the prompt and the generation, are paraphrased. Hence, the semantics of the samples are preserved, while the syntax and word choice differ between D and D_p. For that, the prompt and the generation for each sample s_i are individually paraphrased to retrieve a new sample s_{p_i}, as illustrated in Equation 3.3. As a reminder, each sample is basically a tuple containing an instruction (prompt) and a generation. For simplicity, let us introduce s_{i_P} and s_{i_G} which represent the prompt and the generation of the i-th sample, respectively, such as $s_i = (s_{i_P}, s_{i_G}) = (s_i^{-P_i}, ..., s_i^0, s_i^1, ..., s_i^{G_i})$.

$$s_{p_i} = (s_{p_{i_P}}, s_{p_{i_G}}) = (s_{p_i}^{-P_i}, ..., s_{p_i}^0, s_{p_i}^1, ...s_{p_i}^{G_i}) = (\text{LLM}_{\text{para}}(s_{i_P}), \text{LLM}_{\text{para}}(s_{i_G})) \tag{3.3}$$

As mentioned above, s_{p_i} denotes the paraphrased version of the i-th sample representing a tuple of the paraphrased prompt $s_{p_{i_P}}$ and the paraphrased generation $s_{p_{i_G}}$. The function LLM_{para} in Equation 3.3 basically represents an LLM that is used to paraphrase a sequence of tokens as mentioned above and will be further illustrated in subsection 4.2.2.

Furthermore, another paraphrasing setting is established by using the model under evaluation to generate the output with the paraphrased instruction prompt $s_{p_{i_P}}$ as input. This basically generates another dataset $D_m = \{s_{m_i}\}_{i=1}^{N}$ that is used for further evaluation in this context. As the other samples of the datasets D and D_p, each sample s_{m_i} of D_m represents a tuple of prompt $s_{m_{i_P}}$ and generation $s_{m_{i_G}}$, as shown in Equation 3.4.

$$s_{m_i} = (s_{m_{i_P}}, s_{m_{i_G}}) = (s_{p_i}^{-P_i}, ..., s_{p_i}^0, s_{m_i}^1, ..., s_{m_i}^{G_i}) = (s_{p_{i_P}}, \text{LLM}(s_{p_{i_P}})) \tag{3.4}$$

Technically, $s_{m_{i_P}} = s_{p_{i_P}}$, but for clarity $s_{m_{i_P}}$ will be denoted independently to establish its affiliation to D_m and not be confused with a prompt for a sample in D_p. The function $LLM(\cdot)$ in Equation 3.4 basically returns the generation of the current LLM under evaluation and takes in a sequence of tokens $s_i^{-P_i}, ..., s_i^0$. The dataset D_m is generated to add an additional layer of obfuscation when considering the original D and the paraphrased one D_p.

Given that, the following two settings are established and, for simplicity, will be named accordingly in the following thesis:

- **Paraphrased**: This setting refers to the evaluations regarding the dataset D_p. Samples s_{p_i} in this setting will be called paraphrased samples.

- **Model-Generated**: This setting refers to the evaluation regarding the dataset D_m. Alternatively, in this setting, the samples s_{m_i} will be called model-generated samples.

3.4. Language Model Components

Modern Large Language Models still rely on the Transformer architecture to create appropriate text based on an input prompt. In other words, autoregressive language models, for example, the Generative Pre-Trained Transformer (GPT) model series from OpenAI, predict tokens based on a sequence of input tokens using an adapted decoder-only architecture inspired by the original Transformer [VSP+17].

3.4.1. Different Layers and Components

Hence, we have many trainable parameters from different components. Some of them, for example, activation functions, input layer normalization, post-attention layer normalization, etc., usually included in every Large Language Model, are not trainable and therefore no gradient of the loss function with respect to these components is calculated. For a generic LLM that takes in a sequence of tokens, we might have the following trainable parameters without considering positional encoding, etc.:

- **Input Embedding**: This parameter takes in an already tokenized and encoded sequence and transforms it into the embedding space using, for example, a matrix $\mathbf{E} \in \mathbb{R}^{|V| \times d_x}$.
- **Multiple Transformer Block**: Each transformer block is applied multiple times depending on the model architecture. A generic transformer decoder consists of:
  - **Self-Attention**: Three matrices representing the query, key, and value weights in self-attention:

    $$\mathbf{W}^{(q)} \in \mathbb{R}^{d_x \times d_q}, \mathbf{W}^{(k)} \in \mathbb{R}^{d_x \times d_k}, \mathbf{W}^{(v)} \in \mathbb{R}^{d_x \times d_v}.$$

    [cRS23]
  - **Multiple MLP Projections**: Each transformer block also contains a position-wise feed-forward network (in this example, a two-layer MLP). For every block, the model learns

    $$\mathbf{W}^{(1)} \in \mathbb{R}^{d_x \times d_{ff}}, \qquad \mathbf{b}^{(1)} \in \mathbb{R}^{d_{ff}},$$
    $$\mathbf{W}^{(2)} \in \mathbb{R}^{d_{ff} \times d_x}, \qquad \mathbf{b}^{(2)} \in \mathbb{R}^{d_x},$$

    where d_{ff} is the hidden width of the MLP. It can be written as one liner: $\mathrm{MLP}(x) = \mathbf{W}^{(2)}\big(\mathrm{ReLU}(\mathbf{W}^{(1)}x + \mathbf{b}^{(1)})\big) + \mathbf{b}^{(2)}$, where x is a training sample or in the case of Large Language Models, the output of a previous layer component.

- **Output Embedding**: A final linear layer projects the network output back to vocabulary logits. In some LLMs, the weights are tied to the input embedding [PW17]:

$$\mathbf{W}^{(o)} = \mathbf{E}^\top \in \mathbb{R}^{d_x \times |V|}$$

  where $|V|$ is the size of the vocabulary. (Some models keep an independent $\mathbf{W}^{(o)}$ instead.)

As already mentioned, this is just a simplified illustration of the trainable parameters an LLM could have. In practice, the attention mechanism can be divided into multiple attention heads to capture different semantics in each head. For a detailed illustration of the Transformer decoder, see the original paper "Attention Is All You Need" [VSP+17]. Now that the notion of trainable parameters for Large Language Models is put into context, the original assumption of gradient similarity is established in the following sections.

3.5. Gradient Similarity between Samples

Gradients form the basis of how machine learning models are trained, as they determine the parameter updates during optimization. For a given sample s_a, the gradient $\nabla_\theta \mathcal{L}(s, \theta)$ encodes how the model weights would change if the sample is used for training. Thus, gradients carry information about the influence of a sample on the learning process. To compare this influence across samples s_{p_i} and s_i, the corresponding flattened gradients $\mathbf{v}_\theta(s_{p_i})$ and $\mathbf{v}_\theta(s_i)$ (see subsection 3.5.1 can be compared using a similarity measure $\mathrm{sim}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_i)\big)$ (see subsection 3.5.2. A high similarity indicates that the two samples encourage updates in similar directions, while a low or negative similarity reflects diverging effects on the model parameters.

Although nearly all the equations in the upcoming sections are written explicitly for paraphrased samples $s_{p_i} \in D_p$, the same definitions apply to model-generated samples $s_{m_i} \in D_m$. In both cases, the sample is paired with an original counterpart $s_i \in D$, and the flattened gradients $\mathbf{v}_\theta(s_{m_i})$ can be substituted in place of $\mathbf{v}_\theta(s_{p_i})$ without changing the form of the cosine similarity or accuracy metrics. For clarity, the exposition uses s_{p_i}, but every derivation remains valid if s_{p_i} is replaced by s_{m_i}.

3.5.1. Gradient Flattening

Equation 3.2 illustrates how gradient descent updates the model parameters for training iteration a and the current model parameters θ_a. However, the setting in this work focuses on already fine-tuned models, and hence only the final model parameters θ_T are used for gradient computations. For example, the loss function gradient for a sample s_i and the trained model parameters θ_T w.r.t. the model parameters θ is denoted as $\nabla_\theta \mathcal{L}(s_i, \theta_T)$.

As illustrated in subsection 3.4.1, the model parameter space θ consists of many different variations of matrices with different sizes and dimensions. For simplicity, denote the parameter space θ as a generic set of parameter matrices $\{\mathbf{W}^{(l,k)}\}_{l=1}^{L}$, where $\mathbf{W}^{(l,k)} \in \mathbb{R}^{n_{\mathbf{W}^{(l,k)}} \times d_{\mathbf{W}^{(l,k)}}}$. Here, L denotes the number of layers, and k just the k-th weight in layer l, where $k \in 1, \ldots, K_l$ and K_l is the number of weights inside layer l. Depending on which LLM is used, the number of layers L, the weights inside these layers, and the dimensions of those weights $\mathbb{R}^{n_{\mathbf{W}^{(l,k)}} \times d_{\mathbf{W}^{(l,k)}}}$ differ in size and dimension. The size of the model, in other words, the number of trainable parameters a model has, is shown in Equation 3.5.

$$M = \sum_{l=1}^{L} \sum_{k=1}^{K_l} n_{\mathbf{W}^{(l,k)}} \cdot d_{\mathbf{W}^{(l,k)}} \tag{3.5}$$

As an example, today, there are many different Large Language Models with different sizes, for example, Llama 3.1 consists of 405 billion parameters [DJP+24] and, hence, for Llama 3.1, $M \approx 405,000,000,000$.

For simplicity, call $\nabla_\theta \mathcal{L}(s_i, \theta_T)$ the full model gradient for sample s_i. To compute this full model gradient, the gradients w.r.t. to each model weight $\nabla_{\mathbf{W}^{(l,k)}} \mathcal{L}(s_i, \theta_T) \in \mathbb{R}^{n_{\mathbf{W}^{(l,k)}} \times d_{\mathbf{W}^{(l,k)}}}$ have to be calculated and combined to represent the gradient w.r.t. the whole parameter space. After the computation of each individual weight gradient $\nabla_{\mathbf{W}^{(l,k)}} \mathcal{L}(s_i, \theta_T)$, they are flattened into a gradient vector that can be calculated as:

$$\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i) \;\coloneqq\; \mathrm{vec}_r\big(\nabla_{\mathbf{W}^{(l,k)}} \mathcal{L}(s_i, \theta_T)\big) \;\in\; \mathbb{R}^{n_{\mathbf{W}^{(l,k)}} d_{\mathbf{W}^{(l,k)}}}.$$

The row-major vectorization function $\mathrm{vec}_r(\cdot)$, is the mapping $\mathrm{vec}_r : \mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$, $\quad \mathbf{a} = \mathrm{vec_r}(\mathbf{A})$, where $\mathbf{A} = [A_{ij}] \in \mathbb{R}^{m \times n}$, $\quad i = 0, \ldots, m-1$, $\quad j = 0, \ldots, n-1$. Then this row-major vectorization is explicitly written as follows:

$$\mathbf{a} = \mathrm{vec}_r(\mathbf{A}) = \begin{bmatrix} A_{0,0} \\ A_{0,1} \\ \cdots \\ A_{0,n-1} \\ A_{1,0} \\ A_{1,1} \\ \cdots \\ A_{1,n-1} \\ \vdots \\ A_{m-1,0} \\ \cdots \\ A_{m-1,n-1} \end{bmatrix} \in \mathbb{R}^{mn}.$$

[Cur23]. For example,

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \in \mathbb{R}^{2\times 3}, \quad \text{vec}_r(\mathbf{A}) = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{bmatrix}.$$

To combine all vectorized layer gradients $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)$ for sample s_i, define the collector operator $\text{col}(\cdot)$ as follows:

$$\text{col}(x_1, \ldots, x_m) \coloneqq \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \in \mathbb{R}^{\sum_j d_j}, \quad \text{where } x_j \in \mathbb{R}^{d_j}.$$

With the above properties and operators in mind, the full flattened gradient vector for sample s_i is defined as

$$\mathbf{v}_\theta(s_i) \coloneqq \text{col}\big(\mathbf{v}_{\mathbf{W}^{(0,0)}}(s_i), \mathbf{v}_{\mathbf{W}^{(0,1)}}(s_i), \ldots, \mathbf{v}_{\mathbf{W}^{(L,K_L)}}(s_i)\big) \in \mathbb{R}^M.$$

For better understanding, the whole gradient flattening process covered in this subsection is illustrated in Figure 3.1. As a result of the gradient flattening process and the notion

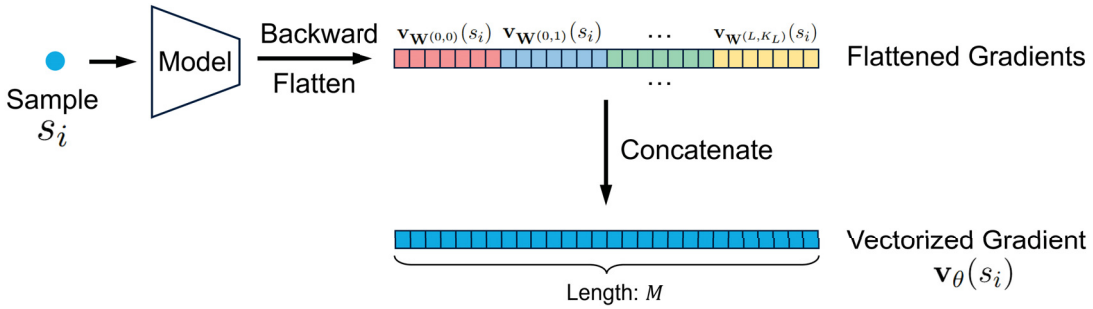


Figure 3.1.: Visual representation of gradient flattening for sample s_i.

of $\mathbf{v}_\theta(s_i)$, these vectorized representations of the gradients for different samples can be compared. In other words, the similarity between these can be calculated, which is elaborated in the following section.

3.5.2. Similarity Measure

Generally, a similarity measure is a real-valued function that quantifies the similarity between two entities. There are many different ways to measure the similarity between two mathematical objects such as vectors. Norms can be used to measure the distance between two points in space, for example, the L_1- and the L_2-norm, called the Manhattan and Euclidean norms, respectively. Another way of measuring the similarity between two vectors is the dot product.

Magnitude versus Direction

The standard dot product between two non-zero vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ is defined as

$$\mathbf{a}^\top \mathbf{b} \;=\; \|\mathbf{a}\|\,\|\mathbf{b}\| \cos\varphi,$$

where φ denotes the angle between them. Consequently, a large dot product can arise in two distinct ways:

1. **Directional alignment**: If φ is small (for example, $\cos\varphi \approx 1$), the vectors point in nearly the same direction and therefore encode similar information.
2. **Magnitude imbalance**: Even when φ is close to $90°$ (poor alignment), the product can still be large if either $\|\mathbf{a}\|$ or $\|\mathbf{b}\|$ is large.

The magnitude of computed gradients from a training set can vary greatly [SWS21] and sometimes blurs the real influence of the gradients in a training set when comparing their local influence on each other [BBD20]. To isolate direction alone, one typically uses cosine similarity,

$$\mathrm{sim}_{\cos}(\mathbf{a}, \mathbf{b}) \;=\; \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|},$$

which normalizes both vectors and depends solely on the angle φ.

Implications for gradient-based influence analysis

Using cosine similarity for gradient comparison has three practical advantages:

1. **Magnitude-invariance**: Two gradients affect the score only through their angle and not by their magnitude.
2. **Sensitivity to local influence**: By basically ignoring large norm outliers, cosine similarity prioritizes gradients whose direction truly matches that of the comparing gradient, which is the criterion for local influence [BBD20].
3. **Interpretability**: A score of $+1$ means perfect alignment, 0 orthogonality, and -1 opposition, which provides an intuitive quantity of how similar the gradients are.

In short, cosine similarity isolates geometrically relevant information (direction) while ignoring the interference factor (magnitude), making it an appropriate choice for gradient-based influence analysis [HL24].

3.6. High Gradient Similarity with Similar Samples

As already mentioned, the original assumption is that when a model is already trained on sample s_i, then in the paraphrased setting

$$\mathrm{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_i)\big) > \mathrm{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_j)\big), \qquad \forall\, j \in \{1, \ldots, N\} \setminus \{i\}. \tag{3.6}$$

That means that the full model gradient of the paraphrased sample s_{p_i} is most similar to its original sample s_i compared to all other original samples s_j. If this is the case, then the gradient-based explanation method is able to find the original data point for its paraphrased equivalent. For both settings, this comparison is made for all samples

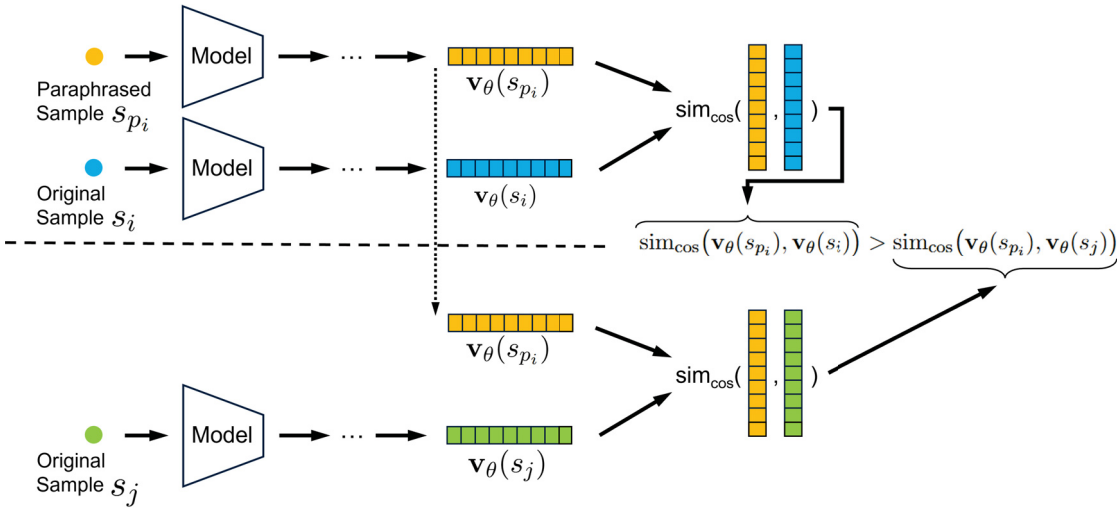


Figure 3.2.: Visual representation of the gradient cosine similarity comparison between the pairs (s_{p_i}, s_i) and (s_{p_i}, s_j).

s_{p_i} and s_{m_i} to calculate a score of how well the model gradients can find the equivalent original data points.

$$\text{accuracy}_\theta(D_p, D) \coloneqq \frac{1}{N}\sum_{i=1}^{N}\left[\!\!\left[\text{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_i)\big) > \max_{\substack{j\in 1,2\ldots,N\\ j\neq i}} \text{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_j)\big)\right]\!\!\right] \tag{3.7}$$

where $[\![P]\!]$ denotes the *Iverson bracket*, defined as

$$[\![P]\!] = \begin{cases} 1, & \text{if the statement } P \text{ is true,} \\ 0, & \text{otherwise.} \end{cases}$$

As shown in Equation 3.7, when computing $\text{accuracy}_\theta(D_p, D) \in [0, 1]$, the gradient for each sample s_i and s_{p_i} has to be calculated. Due to the high-dimensionality of these gradients in the LLM setting, they cannot be cached in the application's memory nor can they be stored on disk, and hence have to be recomputed for each iteration. More about this is mentioned in subsection 4.3.1.

3.6.1. BM25-selected Samples

As identified in subsection 1.3.2, comparing every sample gradient is computationally infeasible. To make the retrieval task tractable, the search space is reduced using a

candidate set of lexically similar samples. This section formally introduces the use of BM25 as the ranking function for this purpose. Instead of comparing $\mathbf{v}_\theta(s_{p_i})$ with all $\mathbf{v}_\theta(s_i),\ \forall s_i \in D$, BM25 is used to find a small subset of lexically similar candidates [RZ09]. Algorithm 1 demonstrates how the b most similar original samples compared to the paraphrased sample s_{p_i} are selected and returned using the index set $\mathcal{C}_i(b)$. If the original sample s_i is not in the final index set $\mathcal{C}_i(b)$, the least similar sample will be overwritten with it to ensure that the original equivalent of s_{p_i} is always present for gradient comparisons. Now that the notion of an index set $\mathcal{C}_i(b)$ with the most lexical overlapping samples to

Algorithm 1: Retrieve the b Most Similar Original Samples via BM25

Data: paraphrased sample s_{p_i}, original dataset $D = \{s_i\}_{i=1}^N$, desired amount b
Result: $\mathcal{C}_i(b)$ — ordered list of the b most similar original samples compared to paraphrased sample s_{p_i}

```
for j ← 1 to N do
    score[j] ← BM25_D(s_{p_i}, s_j)
end
C_i(b) ← argsort_topb(score, b)
    /* The function argsort_topb returns the b indices with the highest
 BM25 scores in descending order and hence, the samples with the most
 lexical overlap according to BM25.    */
// Ensure C_i(b) contains the original sample s_i with index i
if i ∉ C_i(b) then
    C_i(b)[b − 1] ← i // overwrite the least-similar (last) entry
end
return C_i(b)
```

sample s_{p_i} is established, the accuracy in Equation 3.7 can be improved with regards to execution time, as shown in Equation 3.8. Operationally, restricting each comparison to the BM25 candidate set $\mathcal{C}_i(b)$ removes the all-against-all setting: instead of evaluating N cosine similarities per s_{p_i}, only b are evaluated. Consequently, the total cost drops from $\mathcal{O}(N^2)$ to $\mathcal{O}(Nb)$ (with $b \ll N$). Additionally, the overwrite rule guarantees that the connected pair (s_{p_i}, s_i) is always present to always measure the similarity between those counterparts. Therefore, only the gradients for these b candidates per sample s_{p_i} must be computed and loaded, which further reduces runtime and memory footprint.

$$\text{accuracy}_\theta^{(b)}(D_p, D) \coloneqq \frac{1}{N}\sum_{i=1}^{N}\left[\!\!\left[\text{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_i)\big) > \max_{j \in \mathcal{C}_i(b)\setminus\{i\}} \text{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_j)\big)\right]\!\!\right] \tag{3.8}$$

This retrieval accuracy of *BM25* selected samples (see Equation 3.8) is now used in this thesis to measure how well gradient-based explanations are able to find the corresponding original data points for the paraphrased ones. However, as illustrated in subsection 3.4.1, LLMs contain many components that contribute to a full model gradient. Therefore, it makes sense to determine how much each individual layer component contributes to this

retrieval task to determine which ones are the most relevant in this example-based setting.

3.6.2. Layer Component Comparisons

To evaluate how well each individual layer component gradient performs in this retrieval task, the following calculations are employed. Equation 3.8, illustrates the accuracy of the model by selecting samples with BM25 using the full model gradient. To measure how well each individual layer component performs in this setting, this accuracy is adapted to use single layer component gradients, as shown in Equation 3.9.

$$\text{accuracy}^{(b)}_{\mathbf{W}^{(l,k)}}(D_p, D) \coloneqq \frac{1}{N}\sum_{i=1}^{N}\left[\!\!\left[\begin{array}{r} \text{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i}), \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)\big) \\ > \max\limits_{j \in \mathcal{C}_i(b)\setminus\{i\}} \text{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i}), \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)\big) \end{array} \right]\!\!\right] \tag{3.9}$$

As described in section 3.7 below, the cosine similarity of the combined individual layer components will be computed to establish a so-called Greedy-Layer Selection. Care must be taken when combining these component gradients, as the non-linearity of cosine similarity prevents simply summing their individual similarity scores. The following paragraphs illuminate how this problem is addressed accordingly. For simplicity, define the combined gradient vector for two arbitrary layer components $\mathbf{W}^{(l_1,k_1)}$ and $\mathbf{W}^{(l_2,k_2)}$ and sample s_i as

$$\mathbf{v}_{\{\mathbf{W}^{(l_1,k_1)},\mathbf{W}^{(l_2,k_2)}\}}(s_i) \;\coloneqq\; \text{col}\big(\mathbf{v}_{\mathbf{W}^{(l_1,k_1)}}(s_i), \mathbf{v}_{\mathbf{W}^{(l_2,k_2)}}(s_i)\big). \tag{3.10}$$

The cosine similarity is not a linear function due to the norms in the denominator, and therefore

$$\begin{aligned} \text{sim}_{\cos}\big(\mathbf{v}_{\{\mathbf{W}^{(l_1,k_1)},\mathbf{W}^{(l_2,k_2)}\}}(s_{p_i}), \mathbf{v}_{\{\mathbf{W}^{(l_1,k_1)},\mathbf{W}^{(l_2,k_2)}\}}(s_i)\big) \neq \\ \text{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l_1,k_1)}}(s_{p_i}), \mathbf{v}_{\mathbf{W}^{(l_1,k_1)}}(s_i)\big) + \\ \text{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l_2,k_2)}}(s_{p_i}), \mathbf{v}_{\mathbf{W}^{(l_2,k_2)}}(s_i)\big). \end{aligned} \tag{3.11}$$

In other words, Equation 3.11 demonstrates that the cosine similarity of combined gradient vectors is not equal to the sum of its individual gradient vectors due to the non-linearity of the cosine similarity. This can be generalized up to the full model gradient, like

$$\text{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}),\, \mathbf{v}_\theta(s_i)\big) \neq \sum_{l=1}^{L}\sum_{k=1}^{K_l} \text{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i}),\, \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)\big). \tag{3.12}$$

Hence, summing up the cosine similarities for some or all individual layer component gradient vectors for s_{p_i} and s_i does not equal the full model gradient cosine similarity of s_{p_i} and s_i. Therefore, working directly with the cosine similarity does not suit the setting if gradient vectors of multiple layer components need to be combined. However, as shown in the next paragraph, the cosine similarity consists of dot products that are linear.

Linearity of the Dot product for Gradient Vectors

The dot product is *bilinear*, i.e. linear in each of its arguments. For arbitrary vectors $\mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathbb{R}^d$ and scalars $\alpha, \beta \in \mathbb{R}$

$$(\alpha\mathbf{u} + \beta\mathbf{v})^\top \mathbf{w} \;=\; \alpha\,\mathbf{u}^\top \mathbf{w} \;+\; \beta\,\mathbf{v}^\top \mathbf{w}. \tag{3.13}$$

Because the collector operator $\mathrm{col}(\cdot)$ stacks the sub-vectors corresponding to each layer component gradient without overlap and the linearity of the dot product, the dot product between two flattened gradients decomposes into a sum of dot products over the individual components:

$$\mathbf{v}_\theta(s_{p_i})^\top \mathbf{v}_\theta(s_i) \;=\; \sum_{l=1}^{L} \sum_{k=1}^{K_l} \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i). \tag{3.14}$$

Cosine Similarity via Dot Products

Recall that the L_2-norm of any vector can be expressed as the square root of a self-dot product: $\|\mathbf{a}\| = \sqrt{\mathbf{a}^\top \mathbf{a}}$. Consequently, cosine similarity is based on nothing more than dot products. For the *full-model* gradients of the paraphrased sample s_{p_i} and its original counterpart s_i we obtain

$$\begin{aligned} \mathrm{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_i)\big) \;&=\; \frac{\mathbf{v}_\theta(s_{p_i})^\top \mathbf{v}_\theta(s_i)}{\sqrt{\mathbf{v}_\theta(s_{p_i})^\top \mathbf{v}_\theta(s_{p_i})}\,\sqrt{\mathbf{v}_\theta(s_i)^\top \mathbf{v}_\theta(s_i)}} \\ &= \frac{\displaystyle\sum_{l=1}^{L}\sum_{k=1}^{K_l} \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)}{\sqrt{\displaystyle\sum_{l=1}^{L}\sum_{k=1}^{K_l} \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})}\,\sqrt{\displaystyle\sum_{l=1}^{L}\sum_{k=1}^{K_l} \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)}}. \end{aligned} \tag{3.15}$$

The numerator in Equation 3.15 captures the *directional alignment* of the two gradients, while the denominator normalizes by their magnitudes. Consequently, for a single layer component $\mathbf{W}^{(l,k)}$

$$\mathrm{sim}_{\cos}\big(\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i}), \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)\big) \;=\; \frac{\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)}{\sqrt{\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})}\,\sqrt{\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)}}. \tag{3.16}$$

More generally, let $\mathcal{S} \subseteq \big\{\, \mathbf{W}^{(l,k)} \;\big|\; 0 \le l \le L,\; 0 \le k \le K_l \big\}$ denote an arbitrary set of layer components. Collecting the corresponding gradient sub-vectors with $\mathrm{col}(\cdot)$ and applying the same reasoning yields

$$\mathrm{sim}_{\cos}\big(\mathbf{v}_{\mathcal{S}}(s_{p_i}), \mathbf{v}_{\mathcal{S}}(s_i)\big) \;=\; \frac{\mathbf{v}_{\mathcal{S}}(s_{p_i})^\top \mathbf{v}_{\mathcal{S}}(s_i)}{\sqrt{\mathbf{v}_{\mathcal{S}}(s_{p_i})^\top \mathbf{v}_{\mathcal{S}}(s_{p_i})}\sqrt{\mathbf{v}_{\mathcal{S}}(s_i)^\top \mathbf{v}_{\mathcal{S}}(s_i)}}. \tag{3.17}$$

These reformulations show that the cosine similarity can always be evaluated by using dot products. This observation is utilized in the following section 3.7, where the gradients of selected layer components are incrementally combined. Additionally, these intermediate dot products can be easily stored on the disk and later be picked up for further analysis. This is briefly described in subsection 4.3.3.

3.7. Greedy Layer Selection

In addition to analyzing single layer components, this section introduces a greedy selection algorithm that iteratively builds a subset of components whose combined gradients are most informative. The goal is to find a low-dimensional representation that effectively preserves the explanatory power of the full-model gradient. This selection can be guided by two different objectives: either by replicating the full-model's similarity scores or by directly maximizing the task-specific retrieval accuracy.

3.7.1. Prerequisites and Notation

The forward greedy approach relies on a common set of precomputed values and definitions. Let

$$\mathcal{W} \;=\; \big\{\,(l,k)\;\big|\;1 \le l \le L,\; 1 \le k \le K_l\big\}$$

denote the *index set* of all layer components of the model (cf. subsection 3.4.1). For every index pair $(l,k) \in \mathcal{W}$ and every sample pair (i,j) representing a paraphrased sample s_{p_i} and a candidate original s_j, the following per-component *dot products* are precomputed and stored:

$$\begin{aligned}
\delta_{i,j}^{(l,k)} &:= \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j),\\
\delta_{i,i}^{(l,k)} &:= \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i}),\\
\delta_{j,j}^{(l,k)} &:= \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j).
\end{aligned} \tag{3.18}$$

For any subset of selected indices $\mathcal{L} \subseteq \mathcal{W}$, the *accumulated* dot products are defined by summing over the components in that subset:

$$D_{i,j}^{\mathcal{L}} := \sum_{(l,k)\in\mathcal{L}} \delta_{i,j}^{(l,k)}, \qquad D_{i,i}^{\mathcal{L}} := \sum_{(l,k)\in\mathcal{L}} \delta_{i,i}^{(l,k)}, \qquad D_{j,j}^{\mathcal{L}} := \sum_{(l,k)\in\mathcal{L}} \delta_{j,j}^{(l,k)}.$$

Using only these accumulated values, the *reconstructed* cosine similarity between s_{p_i} and s_j for the components in $\mathcal{L}$ is:

$$\hat{\gamma}_{i,j}^{\mathcal{L}} \;:=\; \frac{D_{i,j}^{\mathcal{L}}}{\sqrt{D_{i,i}^{\mathcal{L}}}\sqrt{D_{j,j}^{\mathcal{L}}}}. \tag{3.19}$$

3.7.2. Optimization Objectives

The greedy algorithm iteratively adds the layer component that best improves a given objective score. Two distinct objectives are explored.

Objective 1: Replicating Full-Model Similarity

The first objective is to select a subset of components whose combined gradients best reproduce the cosine similarities of the full model. Let $\Gamma^\theta \in [-1,1]^P$ be the vector that stacks all full-model cosine similarities $\gamma^\theta_{i,j} = \mathrm{sim}_{\cos}\big(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_j)\big)$ for all $P = N \cdot b$ pairs of paraphrased samples and their BM25-selected candidates. Similarly, let $\hat{\Gamma}^{\mathcal{L}}$ be the vector of reconstructed similarities $\hat{\gamma}^{\mathcal{L}}_{i,j}$ for a subset $\mathcal{L}$. The objective is to maximize the cosine similarity between these two vectors:

$$\rho(\mathcal{L}) \;\coloneqq\; \mathrm{sim}_{\cos}\big(\hat{\Gamma}^{\mathcal{L}}, \Gamma^\theta\big). \tag{3.20}$$

A higher $\rho(\mathcal{L})$ expresses more similar scores and, therefore, indicates that the subset $\mathcal{L}$ better captures the overall geometric relationships present in the full-model gradients.

Objective 2: Maximizing Retrieval Accuracy

The second objective is more direct: select components that maximize the final retrieval accuracy. Using the reconstructed similarities from Equation 3.19, the retrieval accuracy for a subset $\mathcal{L}$ is defined as:

$$\mathrm{accuracy}^{(b)}_{\mathcal{L}}(D_p, D) \;\coloneqq\; \frac{1}{N}\sum_{i=1}^{N} \left[\!\!\left[\hat{\gamma}^{\mathcal{L}}_{i,i} > \max_{j \in \mathcal{C}_i(b)\setminus\{i\}} \hat{\gamma}^{\mathcal{L}}_{i,j} \right]\!\!\right]. \tag{3.21}$$

This objective directly optimizes for the end-task performance. The two criteria can favor different components: layers that most closely track the global similarity structure (Objective 1) are not necessarily the same as those that are most decisive for distinguishing the correct original sample within its specific BM25 candidate set (Objective 2).

3.7.3. Forward Greedy Selection Algorithms

Starting from an empty set $\mathcal{L}_0 = \varnothing$, the greedy strategy iteratively builds a sequence of nested subsets $\mathcal{L}_1 \subset \mathcal{L}_2 \subset \ldots$. At each step t, it selects the best component $(l^\star, k^\star)$ from the remaining candidates $\mathcal{W}\setminus\mathcal{L}_{t-1}$ to add to the current set, forming $\mathcal{L}_t = \mathcal{L}_{t-1} \cup \{(l^\star, k^\star)\}$. The "best" component is the one that maximizes the chosen objective function, either $\rho(\mathcal{L}_t)$ or $\mathrm{accuracy}^{(b)}_{\mathcal{L}_t}(D_p, D)$. The following algorithms formalize this process for each objective.

3.7.4. Algorithm Properties

Complexity

Let $L_{\mathrm{tot}} = |\mathcal{W}|$ be the total number of layer components and $P = N \cdot b$ be the number of paraphrased–candidate pairs. Each greedy iteration must scan all remaining components. Inside this loop, updating the accumulators and evaluating the objective score involves $\mathcal{O}(P)$ vector operations. This results in a worst-case complexity of $\mathcal{O}(L^2_{\mathrm{tot}}P)$ for both algorithms. In practice, the algorithms can be stopped early once the objective score

plateaus, but they are run to completion in this analysis to gain full insight into component contributions.

Implementation Details and Variants

For numerical stability, a small constant $\varepsilon > 0$ is added to denominators to prevent division by zero. This is crucial if a gradient vector for a component (or a sum of them) happens to be a zero vector. The implementation relies heavily on vectorized operations to maintain an efficient per-iteration cost.

While the focus of this work is on *forward selection*, alternative greedy strategies exist. *Backward elimination* starts with the full set of components and iteratively removes the least important one. More advanced methods like *beam search* could also be used to explore a wider range of subsets. However, forward selection is computationally efficient as it allows for simple updates to the running accumulators without needing to rematerialize large gradient vectors.

3.7.5. Random Projection as a Baseline

To validate that the Greedy Layer Selection algorithm identifies a genuinely influential subset of layer components, its performance is compared with a random projection baseline. This baseline helps to answer the question: Does the greedy method significantly outperform a random selection of an equivalent number of parameters?

The Johnson-Lindenstrauss (JL) lemma suggests that even a random projection can preserve the geometry of the data, making it a valid baseline [JL84]. In this context, the baseline is constructed by projecting the entire gradient vector into a lower-dimensional space. The full, high-dimensional gradient vector for a sample s_i is denoted by $\mathbf{v}_\theta(s_i) \in \mathbb{R}^M$. A random projection matrix $\mathbf{R} \in \mathbb{R}^{d \times M}$, where $d \ll M$, is used to map this gradient to a lower-dimensional space. The projected gradient vector, $\mathbf{p}(s_i) \in \mathbb{R}^d$, is defined as:

$$\mathbf{p}(s_i) \coloneqq \frac{1}{\sqrt{d}} \mathbf{R}\, \mathbf{v}_\theta(s_i) \tag{3.22}$$

where d is the target dimension of the projection [RR21].

However, due to computational constraints that prevent materializing the full gradient vector $\mathbf{v}_\theta(s_i)$ at once, a layer-wise projection strategy is implemented. For each layer component $\mathbf{W}^{(l,k)}$, a separate random matrix $\mathbf{R}^{(l,k)} \in \mathbb{R}^{d_{l,k} \times |\mathbf{W}^{(l,k)}|}$ is used to project the corresponding layer gradient $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)$, where $d_{l,k}$ is the projection dimension for that layer, proportional to its number of parameters $|\mathbf{W}^{(l,k)}|$. The final low-dimensional representation is obtained by concatenating these individual projections:

$$\mathbf{p}(s_i) \coloneqq \operatorname{col}\left(\frac{1}{\sqrt{d_{0,0}}} \mathbf{R}^{(0,0)} \mathbf{v}_{\mathbf{W}^{(0,0)}}(s_i), \ldots, \frac{1}{\sqrt{d_{L,K_L}}} \mathbf{R}^{(L,K_L)} \mathbf{v}_{\mathbf{W}^{(L,K_L)}}(s_i) \right). \tag{3.23}$$

The cosine similarity is then computed between these resulting down projected gradient vectors $\mathbf{p}(s_i)$ and $\mathbf{p}(s_j)$ for each sample pair.

The comparison is performed as follows:

1. The full-model gradient similarity is computed for each sample pair (s_{p_i}, s_j) to establish a ground truth. This involves calculating the gradient for each sample, flattening the gradient tensors for all layers into a single high-dimensional vector $\mathbf{v}_\theta(s_i)$, and then computing the cosine similarity between these flattened vectors.

2. The Greedy Layer Selection (Algorithm 2) is executed to iteratively build a subset of layer components whose combined gradients best optimize the given objective.

3. A layer-wise random projection of the gradients is performed as a baseline. For each sample, the gradient vector of every individual layer component $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i)$ is projected separately into a lower-dimensional space. These individual projected vectors are then concatenated to form a single, final low-dimensional representation of the full model gradient.

4. Finally, the cosine similarity is calculated between these concatenated, low-dimensional projected gradient vectors for each sample pair. The results of this baseline are then compared against full model gradient scores as well.

Algorithm 2: Forward Greedy Layer Selection by Similarity

Data: Precomputed layer dot products $\{\delta_{i,j}^{(l,k)}, \delta_{i,i}^{(l,k)}, \delta_{j,j}^{(l,k)}\}$
Full-model cosine similarities $\gamma_{i,j}^{\theta}$ for all (i,j)
A fixed ordering of pairs (i,j) giving vectors of length P

Result: Ordered list $\left[((l_1,k_1),\rho_1),\ldots,((l_{|\mathcal{W}|},k_{|\mathcal{W}|}),\rho_{|\mathcal{W}|})\right]$

Flatten all (i,j) into a fixed list and build $\Gamma^{\theta} \in \mathbb{R}^P$.
Normalize Γ^{θ} once: $n_\theta \leftarrow \|\Gamma^{\theta}\|_2$.
Initialize accumulators $D_{ij} \leftarrow \mathbf{0} \in \mathbb{R}^P$, $D_{ii} \leftarrow \mathbf{0} \in \mathbb{R}^P$, $D_{jj} \leftarrow \mathbf{0} \in \mathbb{R}^P$.
$\mathcal{L} \leftarrow \varnothing$, $\mathcal{R} \leftarrow \mathcal{W}$ (remaining indices), $\varepsilon > 0$.
for $t \leftarrow 1$ **to** $|\mathcal{W}|$ **do**
  $\rho^\star \leftarrow -\infty$, $(l^\star,k^\star) \leftarrow$ None.
  foreach $(l,k) \in \mathcal{R}$ **do**
    `// Propose adding index` (l,k)
    $\widetilde{D}_{ij} \leftarrow D_{ij} + \delta_{i,j}^{(l,k)}$
    $\widetilde{D}_{ii} \leftarrow D_{ii} + \delta_{i,i}^{(l,k)}$
    $\widetilde{D}_{jj} \leftarrow D_{jj} + \delta_{j,j}^{(l,k)}$
    `// Vectorized reconstruction of Equation 3.19`
    $\hat{\Gamma} \leftarrow \widetilde{D}_{ij} \oslash \left(\sqrt{\widetilde{D}_{ii}} \odot \sqrt{\widetilde{D}_{jj}} + \varepsilon\right)$
    $n \leftarrow \|\hat{\Gamma}\|_2$
    `// Score of Equation 3.20`
    $\rho \leftarrow \frac{\langle \hat{\Gamma}, \Gamma^{\theta} \rangle}{n \cdot n_\theta + \varepsilon}$
    if $\rho > \rho^\star$ **then**
      $\rho^\star \leftarrow \rho$, $(l^\star,k^\star) \leftarrow (l,k)$
      $D_{ij}^\star \leftarrow \widetilde{D}_{ij}$, $D_{ii}^\star \leftarrow \widetilde{D}_{ii}$, $D_{jj}^\star \leftarrow \widetilde{D}_{jj}$
    end
  end
  `// Commit best index of this round`
  $D_{ij} \leftarrow D_{ij}^\star$, $D_{ii} \leftarrow D_{ii}^\star$, $D_{jj} \leftarrow D_{jj}^\star$
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(l^\star,k^\star)\}$, $\mathcal{R} \leftarrow \mathcal{R} \setminus \{(l^\star,k^\star)\}$
  Store $((l^\star,k^\star),\rho^\star)$ in the output list.
end
return ordered list.

Algorithm 3: Forward Greedy Layer Selection by Accuracy

Data: BM25 candidate index sets $\{\mathcal{C}_i(b)\}_{i=1}^{N}$ (each $\mathcal{C}_i(b)$ contains i)
Precomputed per-component dot products $\{\delta_{i,j}^{(l,k)}, \delta_{i,i}^{(l,k)}, \delta_{j,j}^{(l,k)}\}$
Set of layer indices $\mathcal{W}$, small constant $\varepsilon > 0$

Result: Ordered list $\left[((l_1, k_1), a_1), \ldots, ((l_{|\mathcal{W}|}, k_{|\mathcal{W}|}), a_{|\mathcal{W}|})\right]$ of layers and running accuracies

Initialize accumulators $D_{i,j} \leftarrow 0$, $D_{i,i} \leftarrow 0$, $D_{j,j} \leftarrow 0$ for all pairs (i, j) with $j \in \mathcal{C}_i(b)$.
$\mathcal{L} \leftarrow \varnothing$, $\mathcal{R} \leftarrow \mathcal{W}$ (remaining indices).
for $t \leftarrow 1$ **to** $|\mathcal{W}|$ **do**
  $a^\star \leftarrow -\infty$, $(l^\star, k^\star) \leftarrow$ None, store best temporary accumulators.
  foreach $(l, k) \in \mathcal{R}$ **do**
    `// Propose adding` (l, k) `to the selection`
    $\widetilde{D}_{i,j} \leftarrow D_{i,j} + \delta_{i,j}^{(l,k)}$, $\widetilde{D}_{i,i} \leftarrow D_{i,i} + \delta_{i,i}^{(l,k)}$, $\widetilde{D}_{j,j} \leftarrow D_{j,j} + \delta_{j,j}^{(l,k)}$.
    `// Compute reconstructed similarities` $\hat{\gamma}_{i,j}^{\mathcal{L}\cup\{(l,k)\}}$
    $\hat{\gamma}_{i,j} \leftarrow \widetilde{D}_{i,j} / \left(\sqrt{\widetilde{D}_{i,i}}\sqrt{\widetilde{D}_{j,j}} + \varepsilon\right)$ for all (i, j).
    `// Evaluate accuracy of Equation 3.21`
    $a \leftarrow \frac{1}{N}\sum_{i=1}^{N} [\![\hat{\gamma}_{i,i} > \max_{j \in \mathcal{C}_i(b)\setminus\{i\}} \hat{\gamma}_{i,j}]\!]$.
    if $a > a^\star$ **then**
      $a^\star \leftarrow a$, $(l^\star, k^\star) \leftarrow (l, k)$, save $\widetilde{D}$ as current best.
    end
  end
  `// Commit best index of this round`
  $D_{i,j} \leftarrow$ best saved $\widetilde{D}_{i,j}$, $D_{i,i} \leftarrow$ best saved $\widetilde{D}_{i,i}$, $D_{j,j} \leftarrow$ best saved $\widetilde{D}_{j,j}$.
  $\mathcal{L} \leftarrow \mathcal{L} \cup \{(l^\star, k^\star)\}$, $\mathcal{R} \leftarrow \mathcal{R} \setminus \{(l^\star, k^\star)\}$.
  Store $((l^\star, k^\star), a^\star)$ in the output list.
end
return ordered list.

4. Model, Data, and Infrastructure

This chapter presents the experimental setup for the computations laid out in chapter 3. First, the model and data sources are specified, and the acquisition of third-party artifacts is formalized to ensure reproducibility. The construction of the working datasets is then described, including message formatting and paraphrasing procedures that enable controlled comparisons between the original and transformed samples.

To apply the analysis on modern LLMs, three mechanisms are employed: (i) search-space reduction via lexical pre-selection using BM25, (ii) storage of intermediate dot products in place of full gradients, and (iii) layer-wise random projections. The hardware and scheduling environment are summarized to make resource assumptions explicit and to facilitate replication. The overall goal is to isolate per-instance effects while keeping the pipeline deterministic and verifiable.

4.1. HuggingFace as a General Model and Data Provider

In this work, the *HuggingFace* ecosystem serves as the unified provider for models and datasets. Models are retrieved via *transformers* using `from_pretrained(<model>)` to obtain versioned weights, configuration files, and tokenizer assets; tokenizers are loaded analogously to ensure consistent preprocessing across all runs [WDS+20]. Datasets are accessed through the *datasets* library, which offers streaming, on-disk caching, dataset cards with licensing information, and standardized schemas for robust iteration and filtering [LVdMJ+21].

Relying on a single provider simplifies the end-to-end workflow and reduces nondeterminism. In particular:

- **Versioning and Reproducibility**: Pinning a specific revision guarantees that identical artifacts (weights, configs, tokenizer vocabulary, and chat templates) are resolved in subsequent runs.
- **Cached Artifacts**: Local caching enables offline execution and stable re-use of immutable assets without manual bookkeeping.
- **Introspectable Metadata**: Model cards, configuration files, and tokenizer or chat-template metadata (e.g., `tokenizer.chat_template`) can be read programmatically to construct inputs aligned with each model's pretraining or SFT format.

- **Schema-First Data Access**: Standardized dataset schemas and filtering utilities minimize glue code and keep preprocessing auditable.

Taken together, these properties ensure that preprocessing, evaluation, and analysis operate over a consistent, fully specified set of artifacts, thereby supporting exact reproducibility of all experiments.

4.1.1. OLMo

The OLMo model series, developed by the *Allen Institute for AI*, provides open-source LLMs along with their training data and comprehensive training pipelines [GBW+24]. These features make OLMo models particularly suitable for research focused on explainability. Specifically, OLMo models are pre-trained on the *Dolma* dataset, which comprises approximately three trillion tokens collected from diverse sources [SKB+24]. To adapt these models to specific tasks, the *Open Instruct* framework combined with the *TÜLU* dataset is employed for fine-tuning [IWP+23].

In this work, the focus is on the `AMD-OLMo-1B-SFT` model from AMD, which is derived from the original `OLMo-1B` architecture, as detailed in Table 4.1. This specific model was pre-trained using the *Dolma v1.7* dataset, consisting of 1.3 trillion tokens, and subsequently fine-tuned through supervised training on the *TÜLU v2* dataset [LWM+24]. The model

Parameter size	Number of layers	Number of heads	Hidden size	Context length	Vocabulary Size
1.2B	16	16	2,048	2,048	50,280

Table 4.1.: Model architecture size of `AMD-OLMo-1B-SFT` [LWM+24]

is loaded using *HuggingFace*'s *transformers* function `from_pretrained(<model>)` from the class `AutoModelForCausal`, where `<model>` is the model name (`AMD-OLMo-1B-SFT`). It allows published models with corresponding pre-trained weights to be downloaded and further adapted using *PyTorch*. The complete model architecture when loaded with the *transformers* and printed on the console looks like:

```
OlmoForCausalLM(
  (model): OlmoModel(
    (embed_tokens): Embedding(50304, 2048, padding_idx=1)
    (layers): ModuleList(
      (0-15): 16 x OlmoDecoderLayer(
        (self_attn): OlmoAttention(
          (q_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (k_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (v_proj): Linear(in_features=2048, out_features=2048, bias=False)
          (o_proj): Linear(in_features=2048, out_features=2048, bias=False)
        )
```

```
        (mlp): OlmoMLP(
          (gate_proj): Linear(in_features=2048, out_features=8192, bias=False)
          (up_proj): Linear(in_features=2048, out_features=8192, bias=False)
          (down_proj): Linear(in_features=8192, out_features=2048, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): OlmoLayerNorm()
        (post_attention_layernorm): OlmoLayerNorm()
      )
    )
    (norm): OlmoLayerNorm()
    (rotary_emb): OlmoRotaryEmbedding()
  )
  (lm_head): Linear(in_features=2048, out_features=50304, bias=False)
)
```

Table 4.1 shows a vocabulary size of 50,280, while the actual model output shows a size of 50,304. The authors increased the size to be a multiple of 128 to increase training throughput [GBW+24]. As shown above, the model consists of the following trainable components.

- Input Embedding: $\mathbf{E} \in \mathbb{R}^{|V| \times d_x}$
- 16 Decoder Layers ($l = 1, \ldots, 16$):
  - Attention:
    - Query-Projection: $\mathbf{W}_{\text{Q}}^{(l)} \in \mathbb{R}^{d_x \times d_x}$
    - Key-Projection: $\mathbf{W}_{\text{K}}^{(l)} \in \mathbb{R}^{d_x \times d_x}$
    - Value-Projection: $\mathbf{W}_{\text{V}}^{(l)} \in \mathbb{R}^{d_x \times d_x}$
    - Output-Projection: $\mathbf{W}_{\text{O}}^{(l)} \in \mathbb{R}^{d_x \times d_x}$
  - MLP:
    - Gate-Projection: $\mathbf{W}_{\text{gate}}^{(l)} \in \mathbb{R}^{d_x \times d_{ff}}$
    - Up-Projection: $\mathbf{W}_{\text{up}}^{(l)} \in \mathbb{R}^{d_x \times d_{ff}}$
    - Down-Projection: $\mathbf{W}_{\text{down}}^{(l)} \in \mathbb{R}^{d_{ff} \times d_x}$
- LM Head: $\mathbf{W}^{(o)} \in \mathbb{R}^{d_x \times |V|}$

The dimensionalities $|V|$, d_x, and d_{ff} used in the above weight notation correspond to the architecture parameters, as specified below:

- $|V|$: Vocabulary size (for `AMD-OLMo-1B-SFT`, $|V| = 50{,}304$).
- d_x: Hidden size (for `AMD-OLMo-1B-SFT`, $d_x = 2{,}048$).

- d_{ff}: Feedforward size (for `AMD-OLMo-1B-SFT`, $d_{ff} = 8{,}192$).

The total number of parameters M_H for `AMD-OLMo-1B-SFT` with $L = 16$ decoder layers is:

$$\begin{aligned} M_H = & \underbrace{|V| \cdot d_x}_{\text{Input Embedding}} + \underbrace{L\big(4d_x^2 + 3d_x d_{ff}\big)}_{\text{Decoder Layers}} + \underbrace{d_x \cdot |V|}_{\text{LM Head}} \\ = & \; 2\big(|V| \cdot d_x\big) + L\big(4d_x^2 + 3d_x d_{ff}\big) \\ = & \; 2 \cdot 50{,}304 \cdot 2{,}048 + 16\big(4 \cdot 2{,}048^2 + 3 \cdot 2{,}048 \cdot 8{,}192\big) \\ = & \; 1{,}279{,}787{,}008 \end{aligned}$$

However, this number does not match the model output $M = 1{,}176{,}764{,}416$ when calling the function `model.num_parameters()`. The difference $\Delta_M = M_H - M = 103{,}022{,}592$ happens to be exactly $\Delta_M = |V| \cdot d_x = 50{,}304 \cdot 2{,}048$ which is the number of parameters for the embedding. This indicates some form of weight-tying, and when looking into the model configuration and the model source code [GBW+24], this proves to be true[1]. Hence, the final number of trainable parameters is M since the LM head matrix $\mathbf{W}^{(o)} \in \mathbb{R}^{d_x \times |V|}$ is just a form of transposed input embedding matrix $\mathbf{E} \in \mathbb{R}^{|V| \times d_x}$. In summary, each layer component has a fixed number of trainable parameters as illustrated in Table 4.2.

Component	Shape	Number of Parameters
Input Embedding	$\mathbb{R}^{\lvert V\rvert \times d_x}$	$\lvert V\rvert \cdot d_x = 50{,}304 \cdot 2{,}048 = 103{,}022{,}592$
Attention Projections	$\mathbb{R}^{d_x \times d_x}$	$2{,}048^2 = 4{,}194{,}304$
MLP Projections	$\mathbb{R}^{d_x \times d_{ff}}$; $\mathbb{R}^{d_{ff} \times d_x}$	$2{,}048 \cdot 8{,}192 = 16{,}777{,}216$

Table 4.2.: Trainable parameters in AMD-OLMo-1B-SFT per layer components.

4.1.2. LIMA Dataset

The *LIMA* dataset is a subset of the *TÜLU* dataset, containing $\approx 1,000$ carefully curated prompts and responses across various domains, including STEM fields, alongside manually authored samples [ZLX+23]. Similar to other instruction-tuning datasets, it comprises pairs of prompts and corresponding generations. After the transformations stated section 4.2, it forms the original dataset D. in A prompt typically represents a user's instruction or query, specifying the task or information the model should provide. The model's output, referred to as a generation or response, directly addresses the user's prompt. An illustrative example $(s_{1_P}, s_{1_G}) = s_1 \in D$ from the *LIMA* dataset is provided below. This example shows how much quality and depth the *LIMA* dataset has. In this case, the example is also well-referenced and closely resembles an expert-level response.

User prompt s_{1_P}:

[1] HuggingFace configuration: `https://huggingface.co/amd/AMD-OLMo-1B-SFT/blob/main/config.json#L21`

```
Can brain cells move? By movement I mean long distance migration
(preferably within the brain only).
```

Model generation s_{1_G}:

```
The question is relatively broad and one should take into account that
the brain not only consists of neurons, but also glial cells
(supportive cells) and pre-mitotic neuronal stem cells. Furthermore,
as critical fellow-scientists have indicated, developmental stage is
very important, as the developing embryonic brain is very different
from the adult brain.\nHowever, after sifting through various
publications, the answer to the question is actually remarkably
simple: Yes, brain cells migrate.\nIn  the adult brain glial cells
migrate in the brain (Klämbt, 2009). Glial cells are involved in a
myriad of functions, but a notable example of migrating glial cells
are the oligodendrocytes that migrate relative long distances to
find their target axons onto which they wrap themselves to form the
insulating myelin sheath (Tsai and Miller, 2002).\nNeuronal stem cells
migrate over long distances in response to injury (Imitola et al., 2004)
and they migrate from specific stem-cell locations (e.g., hippocampus
and subventricular zone) to other regions (Clarke, 2003).\nPost-mitotic,
but non-differentiated neurons have been shown to migrate in the adult
brain in fish (Scott et al., 2012), and in mammals and non-human
primates as well (Sawada et al., 2011).\nNot surprisingly, glial cells,
stem cells and neurons also migrate during embryonic development.
Most notably, post-mitotic neurons destined to fulfill peripheral
functions have to migrate over relatively long distances from the
neural crest to their target locations (Neuroscience, 2nd ed,
Neuronal Migration).
```

4.2. Preparing the Data

A small portion of the examples contain multi-turn dialogues with multiple prompts and corresponding generations. These are removed from the dataset to keep a classic prompt/generation single-turn theme for all examples and to make them comparable. When removing those dialogue examples, the filtered dataset still consists of $N = 988$ query/response pairs forming the original dataset D.

4.2.1. User/Assistant Message Format

To ensure consistency and compatibility with modern conversational AI frameworks, each example in the dataset is represented as a list of message objects. Every message specifies both a `role`, indicating whether it originates from the user, the assistant, or optionally the system, and a `content` field containing the corresponding text. As stated previously,

the dialogue examples have been filtered to retain exactly one user-assistant exchange per sample, enforcing a single-turn prompt-generation structure for all pairs.

Roles and Their Purpose

- **system**: An optional message that provides general context or high-level instruction, such as persona or tone guidelines.
- **user**: Represents the prompt, serving as the input for which the model must generate a response.
- **assistant**: Contains the model's generation to the user's prompt; this serves as the target output during both training and evaluation.

Dataset Schema

Each data point is stored in JSON format as a dictionary with a single `messages` key, which holds a list of message objects. A typical single-turn user-assistant interaction is structured as follows.

```
{"messages": [
    { "role": "user", "content": "<user prompt>" },
    { "role": "assistant", "content": "<model generation>" }
]}
```

Listing 4.1: User–assistant message pair

This uniform structure facilitates both supervised fine-tuning and later evaluation using the same message format.

4.2.2. Paraphrasing Samples

This section details the practical implementation of the paraphrasing strategy motivated in subsection 1.3.1 and formally described in section 3.3. As mentioned, the approach operates with two different settings, namely *paraphrased* and *model-generated* paraphrasing levels. For the *paraphrased* case, both the prompt s_{i_P} (user message) and the generation s_{i_G} (assistant message) are altered using *OpenAI GPT-4o-mini* with a seed of 42 and a temperature of 1. The rest of the parameters are set to default values. This process in denoted as $\text{LLM}_{\text{para}}(\cdot)$ in Equation 3.3. The corresponding system prompt is illustrated below.

System prompt for $\text{LLM}_{\text{para}}(\cdot)$:

```
You are a paraphrasing expert who is specialized in rewriting text
(questions, statements, etc.) without altering the content.
Keep in mind, that the meaning must not change after the paraphrasing.
Just output the paraphrased text without any additional information.
```

The *OpenAI* API also follows the message format shown in subsection 4.2.1. Consequently, when paraphrasing the example s_{1_P} from subsection 4.1.2, the *OpenAI* API receives a system-user message pair object containing the previously illustrated system prompt and the text to be rephrased. Hence, all the samples and therefore, each user and assistant message from the filtered *LIMA* dataset are individually sent to *OpenAI* to be paraphrased. For example, the paraphrase of the the user prompt s_{1_P} shown in subsection 4.1.2 looks like follows.

Paraphrased user prompt $\text{LLM}_{\text{para}}(s_{1_P}) = s_{p_{1_P}}$:

```
Are brain cells capable of moving? Specifically, I'm referring to
long-distance migration, ideally occurring within the brain.
```

Additionally, subsection A.3.1 illustrates the paraphrases for the messages shown in subsection 4.1.2. This procedure is then repeated for all samples s_i in D to create the dataset D_p mentioned in section 3.3 and Equation 3.3. To create the dataset D_m for the *model-generated* case, the paraphrased user messages from D_p are input to the model under evaluation (`AMD-OLMo-1B-SFT`). To create a sample $s_{m_i} \in D_m$, a paraphrased user message $s_{p_{i_P}}$ is piped through the model (LLM($\cdot$) to generate a new assistant message $s_{m_{i_G}}$ (generation) as first shown in Equation 3.4 while subsection 4.2.3 shows how this is done on a more technical level.

4.2.3. Prepare Model-generated Data Point

Before the user message can be picked up by the model, some modifications are made to the sample. Usually, the user message is embedded inside a form of tag that signals the model where the user message starts and ends. How these tags are formatted and applied differs from model to model and the corresponding training pipeline. In the case of `AMD-OLMo-1B-SFT`, the user prompt s_{1_P} after formatting looks like:

```
<|user|>\n
Are brain cells capable of moving?
Specifically, I'm referring to long-distance migration,
ideally occurring within the brain.\n
<|assistant|>\n
```

The `<|user|>` tag tells the model where the user message starts. The `<|assistant|>` tag tells the model where the user message ends and the assistant message starts. In the space of HuggingFace, this kind of format is usually called *chat template*. After applying such a template, the text is tokenized and converted to a numerical format by a so-called *tokenizer* to make it readable for the model. For `AMD-OLMo-1B-SFT`, the tokenized and converted user prompt s_{1_P} sequence looks like the following:

```
d
     29, 93, 4537, 49651, 187, 6723, 3998, 1341, 7032, 273,
     4886, 32, 13658, 13, 309, 1353, 14339, 281, 1048, 14,
```

```
19893, 10346, 13, 34243, 12952, 1561, 253, 3998, 15, 187,
29, 93, 515, 5567, 49651, 187
```

Each number represents one token; for example, the value 187 encodes the term `\n`. Such sequences of tokens are then picked up by the model to generate a response (the generation), which also represents a sequence of tokens. This sequence is then decoded using the same *tokenizer* as before and yields the model-generated assistant message $s_{m_{1_G}}$. The corresponding decoded answer for the example shown above is illustrated in subsection A.3.2.

This process is then repeated for each sample from the *original* dataset[2] to generate the *paraphrased* dataset[3] D_p and the *model-generated* dataset[4] D_m correspondingly. These *paraphrased* and *model-generated* examples are stored on disk and picked up when needed to guarantee a level of determinism when operating on this newly created dataset.

4.2.4. Supervised Fine-Tuning Setup

Although no additional Supervised Fine-Tuning (SFT) is performed in this work, all data samples are prepared according to established SFT conventions. This ensures full compatibility with existing model evaluation pipelines and facilitates the subsequent analysis of model behavior from an explainability perspective. Specifically, each sample is formatted as a user–assistant message pair, following the schema described in previous sections.

Open-Instruct

In the context of this work, *Open-Instruct* is mainly utilized to preprocess and structure the data according to established SFT conventions. Specifically, the framework offers utilities for converting raw user–assistant-message pairs such as those from the *LIMA* dataset into the standardized chat template format (with e.g., `<|user|>` and `<|assistant|>` tags) used by models like `AMD-OLMo-1B-SFT`. This ensures consistency between the data preparation procedure and the input expectations of the underlying model architecture.

To preprocess the data for Supervised Fine-Tuning (SFT), the *SFTDatasetProcessor* from *Open-Instruct* is utilized. It allows a whole dataset to be tokenized and prepared by forming the corresponding input and output sequences and naturally integrating into the *HuggingFace* ecosystem for SFT [WID+23].

[2] Original dataset: `https://huggingface.co/datasets/lukashinterleitner/LIMA-original`
[3] Paraphrased dataset: `https://huggingface.co/datasets/lukashinterleitner/LIMA-paraphrased-GPT-4o-mini`
[4] Model-generated dataset: `https://huggingface.co/datasets/lukashinterleitner/LIMA-model-generated-amd-AMD-OLMo-1B-SFT`

4.3. Instance- and Gradient-based Explanation

Having established the theoretical framework for gradient similarity in chapter 3, this section illustrates the practical implementation of this method. The core idea is to leverage the loss function gradient with respect to individual data samples to identify influential training examples. As established in the fine-tuning setting (section 3.2), all gradient computations are performed on single instances (i.e., with a batch size of one) to isolate the influence of individual samples. However, applying this concept to modern Large Language Models introduces significant computational and memory-related challenges. The following subsections shed light on the strategies employed to make this analysis manageable, addressing the high dimensionality of gradients, implementing an efficient candidate selection process, and utilizing intermediate results to manage the computational load.

4.3.1. High Dimensionality

The primary challenge in applying the similarity of the gradient to modern Large Language Models is the dimensionality of the parameter space. For the `AMD-OLMo-1B-SFT` model used in this work, the full model gradient $\mathbf{v}_\theta(s_i)$ for sample s_i, as defined in the gradient flattening process (subsection 3.5.1), is a vector in $\mathbb{R}^M$, where $M = 1{,}176{,}764{,}416$. Storing even a single such gradient vector, assuming 32-bit floating point precision (4 bytes per parameter), would require $1{,}176{,}764{,}416 \cdot 4$ bytes $= 4{,}707{,}057{,}664$ bytes ≈ 4.7 GB of memory or disk space.

A naive approach might involve caching the full gradient vector for each sample in the datasets D and D_p, introduced in section 3.2 and section 3.3, respectively. However, for the filtered *LIMA* dataset with $N = 988$ samples, this would amount to storing nearly $2 \cdot 988 \cdot 4.7$ GB ≈ 9.3 TB of data, which is practically infeasible for most research environments. When adding the model-generated dataset D_m, this increases to ≈ 14 TB of data. Consequently, storing the full gradient vectors for each sample is not a viable strategy, necessitating an alternative that avoids storing these high-dimensional objects altogether. How this problem is approached, will be clarified in the next few sections.

4.3.2. Efficient Candidate Selection with BM25

A naive implementation of the accuracy metric in Equation 3.7 would require computing the cosine similarity between each paraphrased sample gradient $\mathbf{v}_\theta(s_{p_i})$ and every original sample gradient $\mathbf{v}_\theta(s_j)$. For the working dataset of size $N = 988$, this results in a computationally intensive $N \times N$ comparison matrix of nearly one million pairs. However, preliminary analyses on smaller subsets revealed that for a given paraphrased sample s_{p_i}, its original counterpart s_i consistently produced one of the highest similarity scores among all possible pairs. This suggests that the full $N \times N$ comparison is largely redundant.

To drastically reduce execution time, the BM25 ranking function, introduced in subsection 3.6.1, is used to pre-filter the comparison set. As detailed in Algorithm 1, instead of comparing against the entire original dataset D, a small candidate set $\mathcal{C}_i(b)$ for each sampled is first identified, containing the b most lexically similar samples for each s_{p_i}. For all experiments conducted in this work, this candidate size is set to $b = 5$, a value deemed sufficient based on preliminary findings. This targeted approach reduces the number of expensive gradient similarity computations from $988 \times 988 = 976{,}144$ to just $988 \times 5 = 4940$ comparisons per dataset, making the overall accuracy calculation tractable without sacrificing meaningful comparisons due to the lexical similarity in the candidate sets.

4.3.3. Intermediate Results

Even with the reduced comparison set, the challenge of handling enormous gradient vectors during computation remains. One solution lies in recognizing that cosine similarity, as shown in Equations 3.15 through 3.17, is constructed entirely from dot products. Instead of calculating and storing the full, prohibitively large gradient vectors, only the constituent dot products for each layer component are computed and stored.

For each pair of relevant samples (s_{p_i}, s_j) where $j \in \mathcal{C}_i(b)$, the following dot products are calculated:

- $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)$ (*cross dot products*)
- $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})$ (*paraphrased dot products*)
- $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)$ (*original dot products*)

This is done for every layer component $\mathbf{W}^{(l,k)}$ in the model. These scalar values are lightweight and can be efficiently stored on disk. They serve as precomputed intermediate results that can be aggregated later to reconstruct the cosine similarity for any combination of layer components, as required by the Greedy Layer Selection algorithm (section 3.7) and other analysis tasks. This strategy completely circumvents the need to store full gradients, thereby resolving the high-dimensionality bottleneck when storing intermediate results.

All intermediate results are stored as *JSON* files to retain the sample ids for later analysis. The three different dot products (*cross dot products*, etc.) stated above are stored in three different files and later combined accordingly. A shortened example of the *cross dot products* file for `AMD-OLMo-1B-SFT` is illustrated below.


```
{
"lima_0": {
    "lima_0": {
        "model.embed_tokens.weight": 2.129183530807495,
```

```
        "model.layers.0.self_attn.q_proj.weight": 0.08331464231014252,
        "model.layers.0.self_attn.k_proj.weight": 0.12357588112354279,
        ...
    },
    "lima_451": {
        "model.embed_tokens.weight": -0.08331018686294556,
        "model.layers.0.self_attn.q_proj.weight": -0.01116477232426405,
        "model.layers.0.self_attn.k_proj.weight": -0.009237218648195267,
        ...
    },
    ...
},
"lima_1": {
    "lima_1": {
        "model.embed_tokens.weight": 1.2379783391952515,
        "model.layers.0.self_attn.q_proj.weight": 0.027665752917528152,
        ...
    },
    ...
},
...
}
```


4.3.4. Random Projection

The core challenge in applying random projection directly is the prohibitive size of the projection matrix $\mathbf{R}$. For a model with $M \approx 1.18$ billion parameters, a projection to a 1% dimension results in a target dimension of $d = 0.01 \cdot M \approx 11.8$ million. The full projection matrix $\mathbf{R}$ would therefore have dimensions of approximately 11.8 million $\cdot$ 1.18 billion. Storing this matrix with 32-bit precision (4 bytes per element) would require approximately $11.8 \cdot 10^6 \cdot 1.18 \cdot 10^9 \cdot 4$ bytes $= 5.5696 \cdot 10^{16}$ bytes, a size on the order of tens of petabytes, which is computationally infeasible to generate or store in memory. Therefore, a layer-wise projection strategy is adopted, leveraging the fast and efficient `CudaProjector` from the `trak` library [PGI+23]. This approach avoids constructing the full projection matrix at once.

The total projection dimension for each configuration is distributed across the model's layer components proportionally to their size. For a given total projection dimension d_{proj}, the dimension for each individual component (l, k), denoted $d_{(l,k)}$, is calculated as shown in Algorithm 4.

This method ensures that larger layers are projected to correspondingly larger-dimensional spaces, preserving more of their geometric information. The rounding to a multiple of 512 is a practical choice for computational efficiency on modern GPU architectures. The result

Algorithm 4: Proportional Allocation of Projection Dimensions to Layer Components

Data : Parameter counts $P_{(l,k)}$ for all layer components $(l,k) \in \mathcal{W}$; Total parameter count $M = \sum_{(l,k)\in\mathcal{W}} P_{(l,k)}$; Target total projection dimension d_{proj}; Minimal component dimension $d_{\min} \leftarrow 512$; Rounding granularity $g \leftarrow 512$;

Result : Projection dimension $d_{(l,k)}$ assigned to each component, $(l,k) \in \mathcal{W}$

foreach $(l,k) \in \mathcal{W}$ **do**

$p_{(l,k)} \leftarrow \frac{P_{(l,k)}}{M}$; `// Fraction of parameters in component` (l,k)

$d_{\text{raw}} \leftarrow p_{(l,k)} \cdot d_{\text{proj}}$; `// Unrounded allocation`

$d_{\text{rounded}} \leftarrow g \cdot \text{round}\left(\frac{d_{\text{raw}}}{g}\right)$; `// Round to nearest multiple of` g

$d_{(l,k)} \leftarrow \max(d_{\min}, d_{\text{rounded}})$; `// Enforce minimal size`

end

return $d_{(l,k)} \mid (l,k) \in \mathcal{W}$

is a set of projected layer gradients that are concatenated to form the final low-dimensional representation for similarity comparison.

For the benchmark, two random projection configurations were used, with the total target dimensions set to 1% and 5% of the total number of model parameters, effectively leading to 11767808 and 58838016. The random projection matrix is sampled from a *Rademacher* distribution provided by the `trak` library [PGI+23].

4.4. Hardware

To mitigate long-running tasks, all computations were executed on a *Slurm* cluster with a *NVIDIA DGX H100* compute node which consists of 8x NVIDIA H100 Tensor Core GPUs, 2 TB RAM and Dual Intel® Xeon® Platinum 8480C Processors with 112 Cores total, 2.00 GHz (Base), 3.80 GHz (Max Boost).

Before starting the calculations, the dataset is sliced into P_D partitions. Each partition is then assigned to a batch job and queued if necessary. Each batch job got assigned a fixed amount of resources. Hence, each job effectively had access to one NVIDIA H100 Tensor Core GPU, 86 GB RAM, and eight cores.

5. Results

This chapter presents all the results for the algorithms and computations mentioned in the previous chapters. As a brief reminder, the *paraphrased* setting operates with samples from the datasets $s_i \in D$ and $s_{p_i} \in D_p$ (see Equation 3.3) while the *model-generated* setting operates with samples from the dataset $s_i \in D$ and $s_{m_i} \in D_m$ (see Equation 3.4. Concretely, the evaluation checks whether flattened gradients $\mathbf{v}_\theta(s_{p_i})$ (paraphrased) and $\mathbf{v}_\theta(s_{m_i})$ (model-generated) retrieve their corresponding original samples $\mathbf{v}_\theta(s_i)$ via cosine similarity, following Equation 3.6 and the accuracy metric defined in Equation 3.8.

The remainder of the chapter is organized as follows. In section 5.1 baseline retrieval results using the full model gradient are reported while section 5.2 analyzes contributions per component. Additionally, section 5.3 compares single-component scores to full-model scores and section 5.4 evaluates compact surrogates via forward greedy selection against a random-projection baseline. Last but not least, section 5.5 summarizes the cost profile.

5.1. Performance of Gradient Similarity as an Explanation Method

All gradient similarities are computed and reconstructed from stored dot products using Equation 3.14 to represent the cosine similarities for both the paraphrased ($\text{sim}_{\cos}(\mathbf{v}_\theta(s_{p_i}), \mathbf{v}_\theta(s_j))$) and the model-generated ($\text{sim}_{\cos}(\mathbf{v}_\theta(s_{m_i}), \mathbf{v}_\theta(s_j))$) case, $\forall s_{p_i} \in D_p$ and $\forall s_{m_i} \in D_m$. To avoid the comparison of all-pairs $\mathcal{O}(N^2)$, for each paraphrased (or model-generated) sample, $s_j \in \mathcal{C}_i(b)$ is restricted to a BM25 candidate set with $b = 5$ (algorithm 1). The metrics $\text{accuracy}_\theta^{(b)}(D_p, D)$ and $\text{accuracy}_\theta^{(b)}(D_m, D)$ are then evaluated as for the full model gradient in Equation 3.8. As shown in Table 5.1, the gradient similarity

Model	**Paraphrased**	**Model-generated**
AMD-OLMo-1B-SFT	0.993	0.218

Table 5.1.: Score for $\text{accuracy}_\theta^{(b)}(D_p, D)$ (paraphrased) and $\text{accuracy}_\theta^{(b)}(D_m, D)$ (model-generated) cases with the full model gradient.

method is able to find nearly all original samples when queried with the paraphrased samples of D_p. For the model-generated setting, this is very different, as it can only identify approximately one out of five original samples when queried with the dataset D_m. When considering that random guessing would lead to a baseline of 0.2 due to the candidate set having a size of $|\mathcal{C}_i(b)| = 5$ when $b = 5$ and $1 \div 5 = 0.2$, the accuracy is not far off.

In the paraphrased setting, only seven samples are wrongly assigned to their original counterparts. It seems that they are not wrongly assigned by coincidence, they are connected by some patterns. For example, four of them (with the ids `lima_802`, `lima_815`, `lima_832` and `lima_849`) contain instructions like translations as shown for sample s_{839} with id `lima_849`.

User prompt s_{839_P}:

```
translate into English: "Der Zug kommt in Frankfurt pünktlich an.
Kerstin geht sofort nach Hause, aber während sie die Treppen hochsteigt,
bemerkt sie einige eigenartige Dinge: bunte Luftballons, rote Kärtchen
in Herzform, rote Rosen."
```

Model output s_{839_G}:

```
The train arrives at Frankfurt on time. Kerstin goes straight home but,
climbing the stairs, she notices some strange things: coloured balloons,
red cards in the shape of a heart, red roses.
```

Its paraphrased equivalent $s_{p_{839}}$ looks like the following.

Paraphrased user prompt $s_{p_{839_P}}$:

```
The train arrives in Frankfurt on time. Kerstin heads home immediately,
but as she climbs the stairs, she notices some strange things: colorful
balloons, red heart-shaped cards, and red roses."
```

Paraphrased model output $s_{p_{839_G}}$:

```
The train reaches Frankfurt punctually. Kerstin heads directly home,
but while ascending the stairs, she observes some peculiar items: colorful
balloons, heart-shaped red cards, and red roses.
```

As shown in the above examples, the paraphrasing LLM ($\text{LLM}_{\text{para}}(\cdot)$) described in subsection 4.2.2 basically follows the instruction in s_{839_P} and translates the text into English instead of simply paraphrasing the text. Hence, the paraphrase no longer preserves the original *instruction-response* structure: the directive ("translate into English") is removed, the paraphrased user prompt ($s_{p_{839_P}}$) already contains the translated content. This shifts both the token distribution and the gradient geometry for $s_{p_{839}}$, moving $\mathbf{v}_\theta(s_{p_i})$ closer to other English instances than to its own original counterpart $\mathbf{v}_\theta(s_i)$.

5.2. Single Layer Performance

This section quantifies the contribution of *single* layer components to retrieval and similarity. For each parameter matrix $\mathbf{W}^{(l,k)}$, the BM25-restricted retrieval accuracy

$\text{accuracy}^{(b)}_{\mathbf{W}^{(l,k)}}(\cdot, D)$ (Equation 3.9) is evaluated to test whether the gradient of a *single* component suffices to recover the original samples. All quantities are reconstructed from per-component dot products (subsection 4.3.3).

The results are reported for both settings (**paraphrased** and **model-generated**), aggregated across depth and component family (Embedding; Attention $Q/K/V/O$; MLP gate/up/down). The means across layers are shown in Table 5.2, and depth-wise trends are visualized in Figures 5.1 and A.2.

	Layer Component	**Paraphrased**	**Model-generated**
Embedding	Embedding	0.993	0.231
Attention	Query-Projection	0.976	0.255
	Key-Projection	0.906	0.208
	Value-Projection	0.98	0.198
	Output-Projection	0.992	0.245
MLP	Gate-Projection	0.992	0.256
	Up-Projection	0.991	0.247
	Down-Projection	0.993	0.235

Table 5.2.: Mean layer component accuracy for the paraphrased ($\text{accuracy}^{(b)}_{\mathbf{W}^{(l,k)}}(D_p, D)$) and the model-generated ($\text{accuracy}^{(b)}_{\mathbf{W}^{(l,k)}}(D_m, D)$) case.

As shown in Table 5.2, single-component gradients already suffice to recover the original item in the *paraphrased* setting: mean accuracies exceed 0.97 for all components except the key projection (0.906), with embedding, the MLP path (gate/up/down) and the attention output projection matching the full-model score ($\approx$ 0.99). This indicates that paraphrasing preserves gradient directions consistently across layers, so gradients from individual components contain enough information to identify the correct original sample. In contrast, the *model-generated* setting collapses towards the 0.2 random-guess baseline level imposed by b=5: accuracies cluster near baseline, with only the MLP gate at 0.256 and the query projection (0.255) providing a small, repeatable improvement; the value projection falls below baseline (0.198). The embedding matrix reaches 0.231, suggesting that lexical overlap alone is insufficient once responses are re-synthesized (model-generated). Overall, the information that supports correct retrieval is broadly distributed in the paraphrased condition but becomes concentrated mainly in the MLP and Q projections in the model-generated condition, while K and V contribute the least to this retrieval objective. However, as already mentioned, the values shown in Table 5.2 are just mean values, some individual layer components under or over perform with regard to the full model gradient (these values are visualized in the Figures A.2 and 5.1). How size and layer component depth affects these scores, is illustrated in the next two subsections.

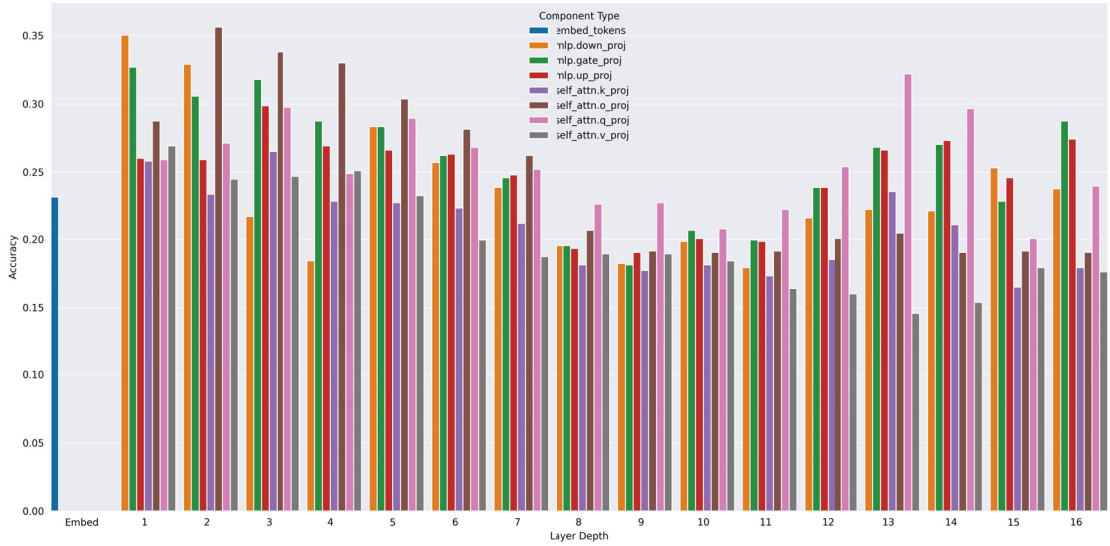


Figure 5.1.: Accuracy per layer in the model-generated setting.

5.2.1. Does Size Matter?

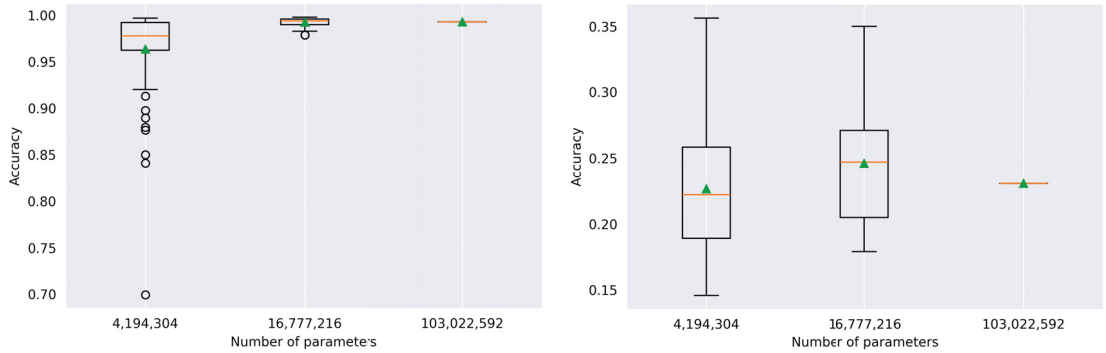
Here, *size* refers to the number of parameters in a component (4,194,304 for the attention projections $Q/K/V/O$, 16,777,216 for the MLP projections gate/up/down, and 103,022,592 for the embedding). Figure 5.2 summarizes retrieval accuracy for these three bins; the sub-panels (Figs. 5.2a and 5.2b) correspond to the *paraphrased* and *model-generated* settings, respectively.

In the *paraphrased* condition (Figure 5.2a), accuracies are essentially at ceiling across all sizes. The 4,194,304 group exhibits a wider spread with several low outliers, whereas the 16,777,216 group is tightly clustered near 0.99 and the 103,022,592 point (embedding) is at 0.993. These differences are small. Because cosine similarity normalizes the vector magnitudes, the slight improvement observed for larger components is best interpreted as reduced variance rather than a substantive size effect.

In the *model-generated* condition (Figure 5.2b), all groups concentrate a little bit above the 0.2 random-guess baseline imposed by b=5. The 16,777,216 group attains a marginally higher median (≈ 0.25) than the 4,194,304 group (≈ 0.23), and the embedding (103,022,592) sits at 0.231. The dispersion is largest for the 4,194,304 group, which bundles both relatively stronger (e.g., Q) and weaker (e.g., V) projections despite identical size. This pattern indicates that *component type*, not parameter count, drives the modest differences: MLP projections outperform on average, whereas K/V lag, even though all four attention projections share the same size.

Overall, there is no meaningful monotonic relationship between parameter count and

retrieval accuracy. In the paraphrased setting, performance is uniformly high regardless of size; in the model-generated setting, accuracy hovers near baseline with only slight advantages for MLP-sized components. The functional role of a component within the architecture matters more than how many parameters it contains.



(a) Paraphrased (b) Model-generated

Figure 5.2.: Component accuracy as a function of parameter count, shown for the (a) **Paraphrased** and (b) **Model-generated** settings. Each boxplot aggregates the accuracy scores for all layer components sharing the same parameter count: Attention projections (4,194,304 parameters), MLP components (16,777,216 parameters), and the single Embedding layer (103,022,592 parameters). In the paraphrased setting, accuracy is consistently high across all component sizes, whereas in the model-generated setting, mean accuracy remains close to the random-guess baseline of 0.2.

5.2.2. Does Depth Matter?

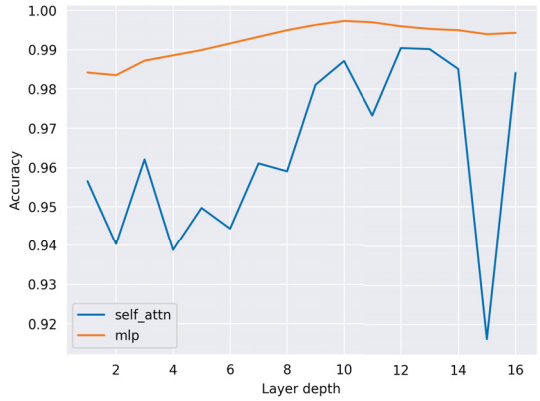
Figure 5.3 reports the retrieval accuracy over layer depth ($l = 1, \ldots, 16$) when matching paraphrased/model-generated samples to their originals using gradient–cosine similarity within the BM25 candidate sets (b=5). Panels 5.3a and 5.3b aggregate over sub-components (Attention vs. MLP), while panels 5.3c - 5.3f shows the accuracy for Attention ($Q/K/V/O$) and MLP (gate/up/down).

In the *paraphrased* condition (left column), accuracies are near ceiling ($\approx$ 0.95–1.00) across most depths and components. MLP slightly but consistently outperforms Attention when averaged over sub-components (panel 5.3a). Within Attention (panel 5.3c), K is the weakest across depth and faces a pronounced drop in the final layer, while Q, V and O remain close to the overall plateau. The accuracy of all three MLP layers increases with layer depth, reaching its high around $l = 10$ followed by a small dip until the last layers.

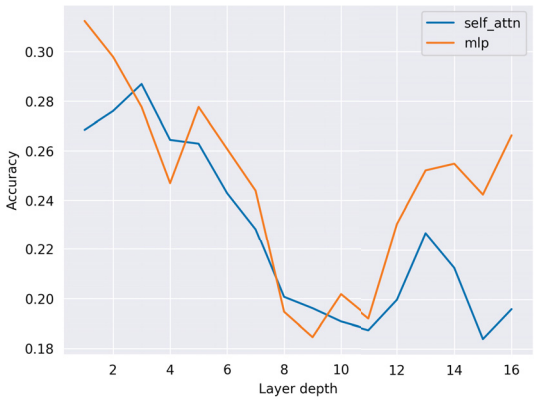
In the *model-generated* condition (right column), accuracies are substantially lower ($\approx$ 0.18–0.33), i.e., only modestly above the random baseline $1/b = 0.20$. Performance is

highest in early layers, dips in the middle, and recovers slightly toward the later layers (panel 5.3b). Aggregated MLP sub-components (panel 5.3f) are pretty similar to Attention overall while MLP's accuracy recovers a little bit better in later layers. Within Attention (panel 5.3d) the ordering $Q, O > K, V$ is nearly consistent across depth.

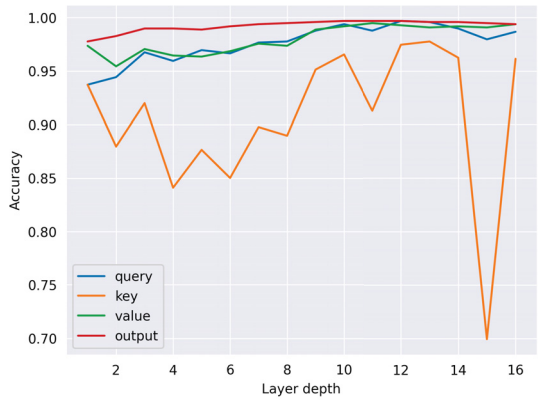
Overall, the *paraphrased* setting preserves gradient directions across layers so that even single-component gradients reliably recover the original sample; by contrast, the *model-generated* setting weakens this alignment, with most retrieval-relevant structure concentrated in the MLP path and the attention Q/O projections, while K/V contribute least to the retrieval objective.



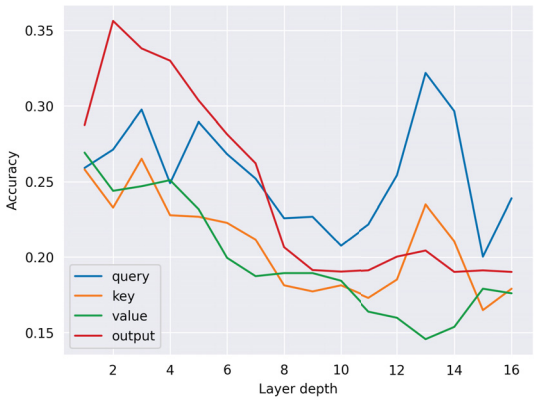
(a) Mean accuracy: Attention vs. MLP (paraphrased).



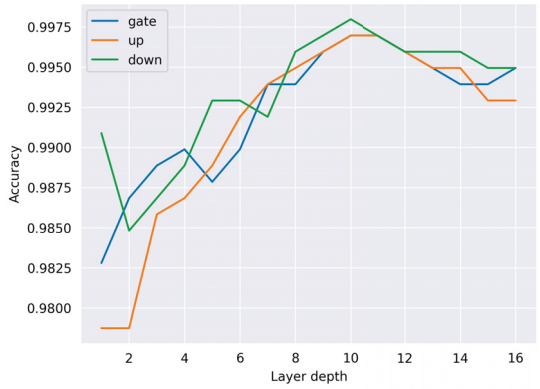
(b) Mean accuracy: Attention vs. MLP (model-generated).



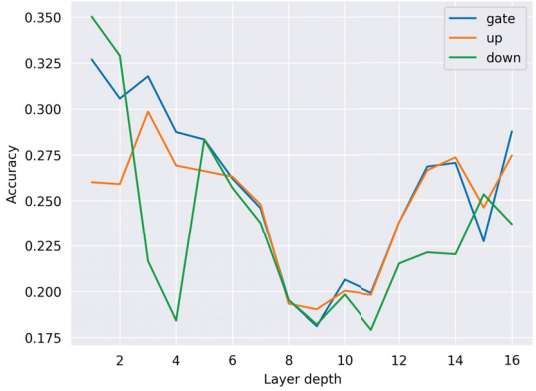
(c) Layer-wise accuracy: Attention sub-components (Q/K/V/O), paraphrased.



(d) Layer-wise accuracy: Attention sub-components (Q/K/V/O), model-generated.



(e) Layer-wise accuracy: MLP sub-components (gate/up/down), paraphrased.



(f) Layer-wise accuracy: MLP sub-components (gate/up/down), model-generated.

Figure 5.3.: Accuracy over layer depth ($l = 1, \ldots, 16$). Left: paraphrased; right: model-generated. Panels (a,b): mean over sub-components; panels (c–f): per-sub-component layer-wise accuracy.

5.3. Comparison of Layer-Gradients and Full-Model-Gradient

With the full-model and single-layer accuracies established for the paraphrased and model-generated settings, the subsequent analysis compares each single-layer component against the full-model gradient. This is done by looking at the cosine-similarity between the different score values $\mathrm{sim}_{\cos}(\hat{\Gamma}^{\mathcal{L}}, \Gamma^{\theta})$ (Equation 3.20).

	Layer Component	**Paraphrased**	**Model-generated**
Embedding	Embedding	0.981	0.275
Attention	Query-Projection	0.969	0.298
	Key-Projection	0.875	0.229
	Value-Projection	0.977	0.615
	Output-Projection	0.988	0.639
MLP	Gate-Projection	0.986	0.536
	Up-Projection	0.988	0.545
	Down-Projection	0.988	0.629

Table 5.3.: Mean cosine similarity between the scores of single layer components ($\hat{\Gamma}^{\mathcal{L}}$) and the full gradient (Γ^{θ}).

Based on Table 5.3, the single components most similar to the full-model gradient in the *paraphrased* setting are the MLP projections and the attention output projection: MLP down/up and Attention output 0.988, MLP gate 0.986; the embedding (0.981) and Attention value (0.977) follow, with Attention query at 0.969 and Attention key lowest at 0.875. In the *model-generated* setting, the highest agreements are Attention output and MLP down (both $\gtrsim 0.63$), then Attention value (0.615) and MLP up (0.545), with MLP gate at 0.536; embedding (0.275) and Attention query (0.298) are much lower, and Attention key is weakest (0.229). Each individual layer alignment with the full model gradient over different layer depths is visualized in the Figures A.3 and A.4.

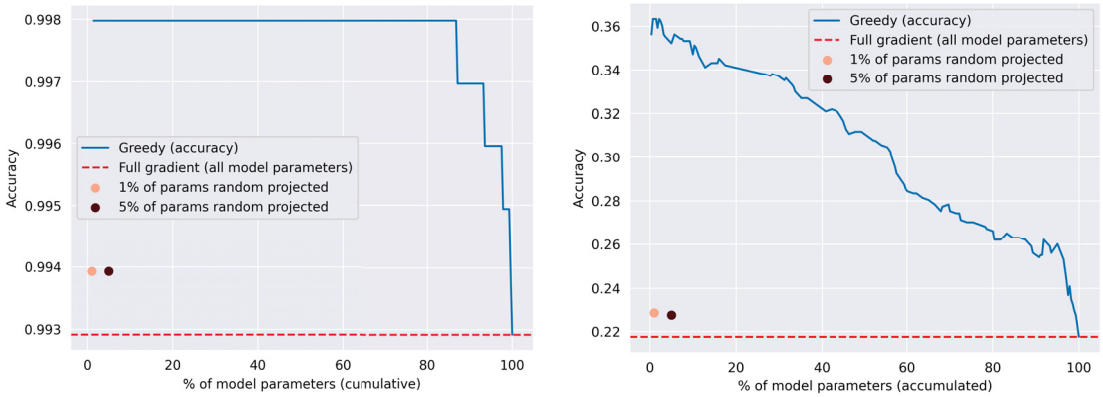
5.4. Greedy-Forward-Layer-Selection vs. Random Projections

Two forward greedy strategies are evaluated using the precomputed dot products: (i) a *similarity-driven* variant (algorithm 2) that maximizes agreement with the full-model score vector Γ^{θ} (objective ρ in Equation 3.20), and (ii) an *accuracy-driven* variant (algorithm 3) that directly maximizes BM25-restricted retrieval accuracy (Equation 3.21). As a geometry-preserving baseline, layer-wise random projections (subsection 3.7.5) are applied at 1% and 5% of M. Budgets on the x-axis are reported as the cumulative fraction of selected model parameters.

5.4.1. Selection by Accuracy

Figure 5.4 shows $\mathrm{accuracy}_{\mathcal{L}}^{(b)}(\cdot, D)$ (Equation 3.21) as layers are added. In the *paraphrased* case (Figure 5.4a), greedy selection reaches near-ceiling accuracy with a very small budget

(≈ 0.998 at $\ll 5\%$ of parameters) and then slightly declines as additional components are appended, ending at the full-model score (0.993). The random-projection baselines at 1% and 5% lie around 0.994, below the early greedy frontier. In the *model-generated* case (Figure 5.4b), accuracy peaks around 0.36 for small budgets and gradually drifts toward ≈ 0.22 as more parameters are added; the 1%/5% random-projection baselines are near the 0.2 candidate-set chance level and are dominated by the greedy curve for small–to–medium budgets. Overall, adding more components does not necessarily improve retrieval, weakly aligned layers can dilute the signal captured by the strongest ones and ultimately approach the accuracy of the full model gradient.



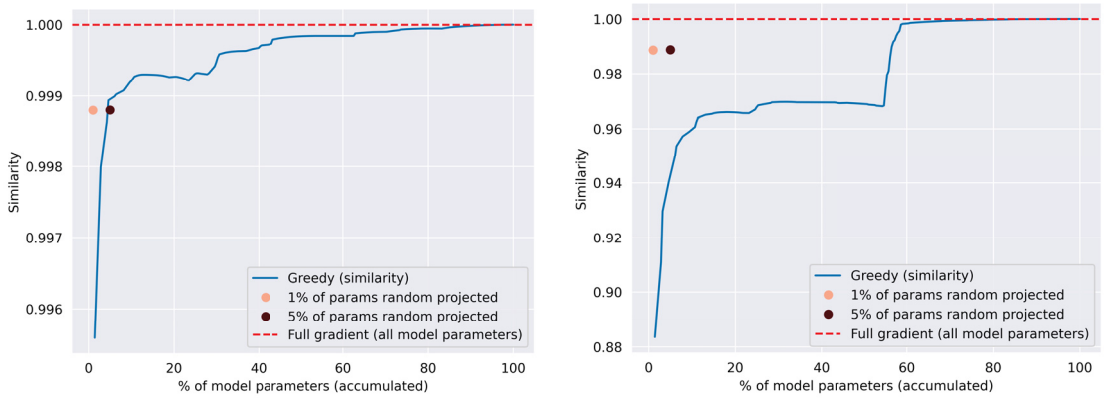
(a) Paraphrased

(b) Model-generated

Figure 5.4.: Greedy Layer Selection and random projection by accuracy.

5.4.2. Selection by Similarity to the Full-Model Gradient

Figure 5.5 tracks $\rho(\mathcal{L}) = \mathrm{sim}_{\cos}(\hat{\Gamma}^{\mathcal{L}}, \Gamma^{\theta})$ while components are added. For *paraphrased* data (Figure 5.5a), similarity jumps to > 0.999 within a tiny budget and monotonically approaches 1.0 as layers accumulate; 1% and 5% random projections land around 0.999 and remain above the greedy trajectory until 5% of parameters. For *model-generated* data (Figure 5.5b), similarity climbs rapidly from ≈ 0.89 to ≈ 0.95 at small budgets, plateaus, and reaches ≈ 1.0 only after a larger fraction of parameters; random projections (1%/5%) also remain above the greedy curve until $\approx 55\%$ of parameters. Together with Figure 5.4, these results highlight a key distinction: maximizing agreement with full-model similarities (high ρ) does not always maximize retrieval accuracy, particularly in the model-generated setting, the accuracy-driven greedy rule can yield smaller, more effective subsets for the retrieval objective.



(a) Paraphrased (b) Model-generated

Figure 5.5.: Greedy Layer Selection compared to random projection by similarity to the full model gradient.

5.5. Execution Time

The execution is done using *Slurm* and time is measured after the model and the tokenizer is loaded. Hence, it contains the time to load the dataset from the disk and perform all computations. Each *Slurm* job logs its execution time separately and saves it on disk for later inspections. To make the results comparable, the individual times from the jobs are summed. Table 5.4 shows the execution times on the hardware described above described in section 4.4. In this context, *dot-products* refers to the computation of the intermediate dot-products mentioned in subsection 4.3.3 and *gradient similarity* refers to the calculation of cosine similarities for the gradients.

As the results in Table 5.4 show, the computation for gradient similarity using random projection (RP) is exceptionally time-consuming. A potential optimization, gradient caching, involves storing the down-projected gradients on disk for reuse. However, this approach was not implemented due to storage limitations. For a complete comparison, a hypothetical execution time for RP with gradient caching was calculated. This estimation, based on the paraphrased setting, suggests the execution time could be reduced from 927.48 hours to approximately 308 hours, assuming negligible read-write overheads from disk.

Computation Type		**Paraphrased (h)**	**Model-generated (h)**
Dot-products		3.54	3.35
Gradient similarity	without RP	4.05	5.39
	with RP	927.48	925.63

Table 5.4.: Execution times (in hours) for dot-products and gradient similarity (with and without Random Projection (RP)) under paraphrased and model-generated settings. The time for RP **with gradient caching** is omitted, as storage limitations prevented its implementation; it was hypothetically estimated to be ∼308 hours for the paraphrased setting, assuming no read-write costs.

6. Discussion

This chapter synthesizes the empirical findings to critically evaluate the *full model gradient* as a baseline for instance-level, gradient-based explanations. The discussion first assesses the suitability of the full gradient as a benchmark across different experimental settings. It then delves into the architectural insights revealed by decomposing the gradient, before addressing the central question of this thesis: whether to select targeted components or project the full gradient when creating compact surrogates. Finally, it outlines the limitations of this work and suggests avenues for future research.

6.1. Evaluating the Full Model Gradient as a Baseline

The empirical results demonstrate that the suitability of the full model gradient as a baseline is highly context-dependent, varying significantly with the nature of the input transformation.

6.1.1. In the Paraphrased Setting: A Performance Ceiling

In the *paraphrased setting*, where input semantics are preserved, the full-model gradient serves as a robust **performance ceiling**. It achieves near-perfect retrieval accuracy of 0.993 within the BM25-selected candidate set (Table 5.1). However, this high performance is not unique to the full gradient. Strikingly, individual architectural components, particularly MLP projections and the attention output projection, attain comparable, and sometimes superior, accuracy on their own (Table 5.2). The cosine similarities of scores induced by these single components also show exceptionally high alignment with the full gradient (means $\gtrsim 0.98$, see Table 5.3).

This indicates that under paraphrasing, the gradient direction relevant for instance identification is stable and broadly distributed across the model's architecture. The key question in this context is not whether a surrogate can match the baseline, but rather how efficiently, in terms of parameter budget and computational cost, it can reach this performance ceiling.

6.1.2. In the Model-Generated Setting: A Weak and Misleading Benchmark

In the *model-generated setting*, where the model re-synthesizes a response, the full gradient's performance collapses, making it a **weak and potentially misleading baseline**. Its retrieval accuracy plummets to 0.218, only marginally better than the random-guess baseline of 0.2 (Table 5.1). The signal of the full gradient appears diluted by weaker layer

components, as even small, greedily-selected parameter subsets consistently outperform it.

Crucially, a surrogate's ability to preserve the global geometry of the full-gradient similarity space is not a sufficient condition for high retrieval performance. For instance, components that best approximate the full gradient's similarity structure (e.g., attention output and MLP down projections, with $\rho \gtrsim 0.63$) do not necessarily yield the highest retrieval accuracy (Table 5.2). Therefore, evaluating surrogates in this setting requires prioritizing task-aligned metrics like retrieval accuracy over geometric alignment to the full gradient.

6.2. Architectural Insights from Gradient Decomposition

The layer-wise analysis provides novel insights into how instance-specific information is encoded in the gradients of the Transformer architecture.

6.2.1. Asymmetric Contribution of Layer Components

A consistent finding is that different architectural components contribute asymmetrically to the retrieval task. The MLP blocks and the attention Q and O projections consistently carry more task-relevant information than the K and V projections. This highlights the significance of a component's **functional role** over its parameter count in determining its explanatory power.

6.2.2. Non-Monotonicity of Performance with Dimensionality

The greedy selection experiments reveal a non-monotonic relationship between the number of included parameters and retrieval accuracy. For both settings, a small, optimally selected subset of layers can achieve higher accuracy than the full model gradient (Figure 5.4). As more, less-informative layers are added, performance degrades, eventually converging toward the full gradient's baseline score. This suggests that adding more components can introduce noise that harms performance on the targeted retrieval task.

6.2.3. Divergent Depth Profiles

The distribution of influential gradients across model depth differs markedly between the two settings. In the paraphrased setting, accuracy is high and relatively stable across all layers. In contrast, in the model-generated setting, performance is strongest in the initial and final layers, with a noticeable dip in the middle layers (Figure 5.3b). This suggests that early-layer features (closer to input) and late-layer features (closer to output) retain some alignment with the original sample, while intermediate layers, which perform more abstract transformations, diverge more significantly.

6.3. Implications: To Select or to Project?

The findings provide a clear, task-dependent answer to the central question of whether to select or project components when building a low-dimensional representation of the gradient.

6.3.1. For Maximizing Retrieval Accuracy: Select

If the primary objective is to build a compact representation that maximizes retrieval accuracy, then **targeted selection is the superior strategy**. The greedy selection algorithm, particularly when optimized directly for accuracy, identifies a small subset of components (typically from MLP and attention Q/O blocks) that outperforms both the full model gradient and random projections at comparable parameter budgets. Furthermore, the selection approach is vastly more computationally efficient. The pre-computation and greedy search completed in hours, whereas the random projection baseline required several hundred hours for the same dataset (Table 5.4). For practical XAI applications where instance-based explanations are framed as a retrieval problem, targeted selection is more effective and scalable.

6.3.2. For Preserving Global Gradient Geometry: Project

If the objective is to create a low-dimensional surrogate that best preserves the geometric relationships of the full-gradient space, then **projection is a highly effective, albeit costly, method**. In line with the Johnson-Lindenstrauss lemma, random projections maintain high fidelity to the full gradient's similarity structure (ρ) even at very low dimensions (Figure 5.5). If the goal is a compressed representation for tasks that depend on this overall geometry, projection is a viable approach, provided the substantial computational overhead is acceptable.

6.4. Limitations and Future Research Directions

The conclusions of this work should be considered in light of its limitations, each of which opens avenues for future research.

- **Model and Data Specificity:** The experiments were conducted on a single 1B-parameter model (`AMD-OLMo-1B-SFT`) and one high-quality instruction-tuning dataset (LIMA). The extent to which these findings generalize to different model architectures, larger model scales, diverse datasets, and other tasks remains an open question. Future work should replicate this analysis across a wider range of models and data domains.
- **Static Analysis:** The analysis is static, using gradients computed at a single, final model checkpoint. Dynamic methods that track influence throughout the training process, as discussed in Dynamic, Gradient-Based Influence Estimation, could provide a more complete picture of how training data shape model behavior.

7. Conclusion

This thesis systematically evaluates low-dimensional representations for instance-based explanations, addressing the central question of whether to *select* architectural components or *project* the full gradient. The findings demonstrate a clear and practical answer: for the goal of retrieving influential training examples, a targeted selection of a small subset of gradient components is both more accurate and computationally efficient than projection. The results pinpoint these influential components as originating primarily from the MLP and attention query/output blocks.

This work also challenges the role of the full model gradient as a default baseline. The analysis reveals it as a less effective approach for instance-based explanation, showing that smaller, architecturally-informed representations can provide a clearer and more discriminative signal. While projection offers remarkable fidelity in preserving the global geometry of the full-gradient space, its prohibitive runtime makes it less suitable for practical applications. Consequently, the findings advocate for choosing small, architecturally aware representations over geometry-preserving projections when building efficient and effective instance-based explanation tools.

Bibliography

[AAE+23] Sajid Ali, Tamer Abuhmed, Shaker H. Ali El-Sappagh, Khan Muhammad, Jose Maria Alonso-Moral, Roberto Confalonieri, Riccardo Guidotti, Javier Del Ser, Natalia Díaz Rodríguez, and Francisco Herrera. Explainable artificial intelligence (XAI): what we know and what is left to attain trustworthy artificial intelligence. *Inf. Fusion*, 99:101805, 2023.

[ABL+22] Ekin Akyürek, Tolga Bolukbasi, Frederick Liu, Binbin Xiong, Ian Tenney, Jacob Andreas, and Kelvin Guu. Towards tracing knowledge in language models back to the training data. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 2429–2446. Association for Computational Linguistics, 2022.

[AI24] Meta AI. Introducing llama 3.1: Our most capable models to date. `https://ai.meta.com/blog/meta-llama-3-1/`, July 2024. Accessed: 10-01-2025.

[BBD20] Elnaz Barshan, Marc-Etienne Brunet, and Gintare Karolina Dziugaite. Relatif: Identifying explanatory training examples via relative influence. *CoRR*, abs/2003.11630, 2020.

[BMR+20] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[CCWB21] Roberto Confalonieri, Ludovik Coba, Benedikt Wagner, and Tarek R. Besold. A historical perspective of explainable artificial intelligence. *WIREs Data Mining Knowl. Discov.*, 11(1), 2021.

[cRS23] Erion Çano, Benjamin Roth, and Andreas Stephan. Lecture notes in deep learning for natural language processing, February 2023.

[Cur23] Vitor Probst Curtarelli. On the generalized vectorization and its inverse. *CoRR*, abs/2308.07928, 2023.

[DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[DJP+24] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[Eur16] European Commission. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance), 2016.

[GBA+23] Roger B. Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez,

Evan Hubinger, Kamile Lukosiute, Karina Nguyen, Nicholas Joseph, Sam McCandlish, Jared Kaplan, and Samuel R. Bowman. Studying large language model generalization with influence functions. *CoRR*, abs/2308.03296, 2023.

[GBW+24] Dirk Groeneveld, Iz Beltagy, Evan Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. Olmo: Accelerating the science of language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15789–15809. Association for Computational Linguistics, 2024.

[GRH+21] Han Guo, Nazneen Rajani, Peter Hase, Mohit Bansal, and Caiming Xiong. Fastif: Scalable influence functions for efficient model interpretation and debugging. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 10333–10350. Association for Computational Linguistics, 2021.

[HL24] Zayd Hammoudeh and Daniel Lowd. Training data influence analysis and estimation: a survey. *Mach. Learn.*, 113(5):2351–2403, 2024.

[IWP+23] Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew E. Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. Camels in a changing climate: Enhancing LM adaptation with tulu 2. *CoRR*, abs/2311.10702, 2023.

[JJZS20] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits. Is BERT really robust? A strong baseline for natural language attack on text classification and entailment. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8018–8025. AAAI Press, 2020.

[JL84] William Johnson and Joram Lindenstrauss. Extensions of lipschitz maps into a hilbert space. *Contemporary Mathematics*, 26:189–206, 01 1984.

[JSS19] Ganesh Jawahar, Benoît Sagot, and Djamé Seddah. What does BERT learn about the structure of language? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 3651–3657. Association for Computational Linguistics, 2019.

[KL17] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1885–1894. PMLR, 2017.

[KWWZ23] Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. DataInf: Efficiently Estimating Data Influence in LoRA-tuned LLMs and Diffusion Models. In *The Twelfth International Conference on Learning Representations*, October 2023.

[LH19] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[LLXZ24] Huawei Lin, Jikai Long, Zhaozhuo Xu, and Weijie Zhao. Token-wise influential training data retrieval for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 841–860. Association for Computational Linguistics, 2024.

[LVdMJ+21] Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[LWM+24] Jiang Liu, Jialian Wu, Prakamya Mishra, Zicheng Liu, Sudhanshu Ranjan, Pratik Prabhanjan Brahma, Yusheng Su, Gowtham Ramesh, Peng Sun, Zhe Li, Dong Li, Lu Tian, and Emad Barsoum. Amd-olmo: A series of

1b language models trained from scratch by amd on amd instinct™ mi250 gpus., October 2024.

[Ope23] OpenAI. Language models can explain neurons in language models. `https://openai.com/index/language-models-can-explain-neurons-in-language-models`, May 2023. Accessed: 09-01-2025.

[PGI+23] Sung Min Park, Kristian Georgiev, Andrew Ilyas, Guillaume Leclerc, and Aleksander Madry. TRAK: attributing model behavior at scale. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 27074–27113. PMLR, 2023.

[PLKS20] Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[PW17] Ofir Press and Lior Wolf. Using the output embedding to improve language models. In Mirella Lapata, Phil Blunsom, and Alexander Koller, editors, *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pages 157–163. Association for Computational Linguistics, 2017.

[RNSS18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. `https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf`, 2018. Accessed: 10-01-2025.

[RR21] Beheshteh T. Rakhshan and Guillaume Rabusseau. Rademacher random projections with tensor networks. *CoRR*, abs/2110.13970, 2021.

[RZ09] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009.

[SKB+24] Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Raghavi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters,

Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: an open corpus of three trillion tokens for language model pretraining research. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 15725–15788. Association for Computational Linguistics, 2024.

[SWS21] Yi Sui, Ga Wu, and Scott Sanner. Representer point selection via local jacobian expansion for post-hoc classifier explanation of deep neural networks and ensemble models. In Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan, editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 23347–23358, 2021.

[SZVS22] Andrea Schioppa, Polina Zablotskaia, David Vilar, and Artem Sokolov. Scaling up influence functions. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 8179–8186. AAAI Press, 2022.

[TDP19] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4593–4601. Association for Computational Linguistics, 2019.

[TMS+23] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen

Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017.

[WDS+20] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, pages 38–45. Association for Computational Linguistics, 2020.

[WID+23] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Raghavi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. How far can camels go? exploring the state of instruction tuning on open resources. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

[WMSJ25] Jiachen T. Wang, Prateek Mittal, Dawn Song, and Ruoxi Jia. Data shapley in one training run. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025.

[WSM+19] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[WXL+22] Boxin Wang, Chejian Xu, Xiangyu Liu, Yu Cheng, and Bo Li. Semattack: Natural textual attacks via different semantic spaces. In Marine Carpuat,

Marie-Catherine de Marneffe, and Iván Vladimir Meza Ruíz, editors, *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 176–205. Association for Computational Linguistics, 2022.

[XLZ+24] Haoyi Xiong, Xuhong Li, Xiaofei Zhang, Jiamin Chen, Xinhao Sun, Yuchen Li, Zeyi Sun, and Mengnan Du. Towards explainable artificial intelligence (XAI): A data mining perspective. *CoRR*, abs/2401.04374, 2024.

[YLR+20] Yang You, Jing Li, Sashank J. Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[YTS+22] Chih-Kuan Yeh, Ankur Taly, Mukund Sundararajan, Frederick Liu, and Pradeep Ravikumar. First is better than last for language data influence. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022.

[ZCY+24] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. Explainability for large language models: A survey. *ACM Trans. Intell. Syst. Technol.*, 15(2):20:1–20:38, 2024.

[ZLX+23] Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: less is more for alignment. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023.

ChatGPT, Gemini and Writefull were used to rephrase sentences or words to make the language, tonality, and writing style more coherent and easier to read.

Acronyms

AI Artificial Intelligence. 1, 2

DNN Deep Neural Network. 1

EK-FAC Eigenvalue-corrected Kronecker-Factored Approximate Curvature. 9

EU European Union. 1

GDPR General Data Protection Regulation. 1

GPT Generative Pre-Trained Transformer. 16

JL Johnson-Lindenstrauss. 27

k-NN k-Nearest Neighbors. 9

LLM Large Language Model. 1–3, 7, 9–11, 14–18, 21, 22, 31, 32, 39, 44

LM Language Model. 8, 33, 34

LoRA Low-rank adaptation. 9

MLP Multilayer Perceptron. 4, 9, 11, 33, 34, 45–50, 55–57, 59

RP Random Projection. xi, 53

SFT Supervised Fine-Tuning. 31, 38

XAI Explainable Artificial Intelligence. 1, 2

A. Appendix

A.1. Nomenclature

The following tables define the key symbols and notations used throughout this thesis for clarity and reference.

Symbol	Meaning
N	Number of samples in the (filtered) original dataset D
i, j	Sample indices, i for (para-/model-generated) query, j for original candidate
a	Training iteration index
$l \in \{1, \dots, L\}$	Layer index
$k \in \{1, \dots, K_l\}$	Component index within layer l
b	Size of the BM25 candidate set $\mathcal{C}_i(b)$
$P = N \cdot b$	#pairs (paraphrased $\times$ BM25 candidates) used in evaluation
θ	Model parameters
θ_0	Model parameters at the start of training
θ_a	Model parameters at iteration a
θ_T	Model parameters at the end of training (fine-tuned model)
η_a	Learning rate at iteration a

Table A.1.: General indices and counts.

Symbol	Meaning
$D = \{s_i\}_{i=1}^N$	Original dataset (instruction–response pairs)
$D_p = \{s_{p_i}\}_{i=1}^N$	Paraphrased dataset (prompt and response paraphrased)
$D_m = \{s_{m_i}\}_{i=1}^N$	Model-generated dataset (paraphrased prompts, model outputs)
$s_i = (s_{i_P}, s_{i_G})$	Sample i: prompt s_{i_P} and generation s_{i_G}
$s_{i_P} = s_i^{-P_i}, \dots, s_i^0$	Prompt tokens of length P_i
$s_{i_G} = s_i^1, \dots, s_i^{G_i}$	Generation tokens of length G_i
$s_{p_i} = (s_{p_{i_P}}, s_{p_{i_G}})$	Paraphrased sample
$s_{m_i} = (s_{m_{i_P}}, s_{m_{i_G}})$	Model-generated sample ($s_{m_{i_P}} = s_{p_{i_P}}$)
$\mathrm{LLM}_{\mathrm{para}}(\cdot)$	Paraphrasing model/function
$\mathrm{LLM}(\cdot)$	Model under evaluation (produces $s_{m_{i_G}}$)

Table A.2.: Datasets, samples, and tokenization.

Symbol	Meaning
$\mathcal{L}(s_i^j, \theta)$	Token-level loss at token j of sample i
$\mathcal{L}(s_i, \theta) = \frac{1}{G_i} \sum_{j=1}^{G_i} \mathcal{L}(s_i^j, \theta)$	Per-sample loss
$p_\theta(s_i^j \mid s_i^{-P_i}, \ldots, s_i^{j-1})$	Next-token probability under the model
$\nabla_\theta \mathcal{L}(s_i, \theta)$	Gradient of loss w.r.t. θ for sample s_i
$\theta_{a+1} = \theta_a - \eta_a \nabla_\theta \mathcal{L}(s_a, \theta_a)$	Gradient descent update

Table A.3.: Loss and training notation.

Symbol	Meaning
$\mathcal{W} = \{(l, k)\}$	Index set of all layer components
$\mathbf{W}^{(l,k)}$	Parameter matrix of component k in layer l
L	Number of decoder layers
K_l	Number of components in layer l
M	Total trainable parameters
$\mathbf{E} \in \mathbb{R}^{\lvert V \rvert \times d_x}$	Input embedding matrix
$\mathbf{W}^{(o)} = \mathbf{E}^\top \in \mathbb{R}^{d_x \times \lvert V \rvert}$	Tied LM head (output projection)
$\lvert V \rvert$	Vocabulary size
d_x	Hidden size
d_{ff}	Feed-forward (MLP) width
$\mathbf{W}_{\mathrm{Q}}^{(l)}, \mathbf{W}_{\mathrm{K}}^{(l)}, \mathbf{W}_{\mathrm{V}}^{(l)}, \mathbf{W}_{\mathrm{O}}^{(l)}$	Attention Q/K/V/O projections in layer l
$\mathbf{W}_{\mathrm{gate}}^{(l)}, \mathbf{W}_{\mathrm{up}}^{(l)}, \mathbf{W}_{\mathrm{down}}^{(l)}$	MLP gate/up/down projections in layer l
M_H	Naive param count without weight-tying
Δ_M	Difference due to tying ($\Delta_M = \lvert V \rvert \cdot d_x$ here)

Table A.4.: Model/parameter notation.

Symbol	Meaning
$\mathrm{vec}_r(\cdot)$	Row-major vectorization $\mathbb{R}^{m \times n} \to \mathbb{R}^{mn}$
$\mathrm{col}(x_1, \ldots, x_m)$	Collector/concatenation: stack vectors vertically
$\odot,\ \oslash$	Elementwise product and elementwise division
$\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_i) = \mathrm{vec}_r\left(\nabla_{\mathbf{W}^{(l,k)}} \mathcal{L}(s_i, \theta_T)\right)$	Flattened gradient for component (l, k)
$\mathbf{v}_\theta(s_i) = \mathrm{col}\left(\mathbf{v}_{\mathbf{W}^{(0,0)}}(s_i), \ldots, \mathbf{v}_{\mathbf{W}^{(L,K_L)}}(s_i)\right)$	Full flattened gradient for s_i

Table A.5.: Operators and flattened gradient vectors.

Symbol	Meaning
$\mathbf{R} \in \mathbb{R}^{d \times M}$	Random projection matrix (Rademacher) for full vector (conceptual)
$\mathbf{R}^{(l,k)} \in \mathbb{R}^{d_{(l,k)} \times \lvert \mathbf{W}^{(l,k)} \rvert}$	Per-component projector (used in practice)
$d,\ d_{(l,k)}$	Projection dimensions (global / per component)
d_{proj}	Target total projection dimension (budget)
$d_{\min},\ g$	Min per-component dim and rounding granularity
$P_{(l,k)}$	Parameter count of component (l, k)
$\mathbf{p}(s_i)$	Concatenated projected gradient for sample s_i

Table A.9.: Random projection baseline notation.

Symbol	Meaning
$\langle \mathbf{a}, \mathbf{b} \rangle = \mathbf{a}^\top \mathbf{b}$	Dot product
$\mathrm{sim}_{\cos}(\mathbf{a}, \mathbf{b}) = \dfrac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}$	Cosine similarity
$\gamma^{\theta}_{i,j} = \mathrm{sim}_{\cos}\left(\mathbf{v}_\theta(\, s_{p_i}\,), \mathbf{v}_\theta(\, s_j\,)\right)$	Full-model similarity (paraphrased $\rightarrow$ original)
$\Gamma^{\theta} \in \mathbb{R}^P$	Vector stacking all $\gamma^{\theta}_{i,j}$ over evaluated pairs
$\delta^{(l,k)}_{i,j}$	Per-component cross dot: $\mathbf{v}_{\mathbf{W}^{(l,k)}}(s_{p_i})^\top \mathbf{v}_{\mathbf{W}^{(l,k)}}(s_j)$
$\delta^{(l,k)}_{i,i}$, $\delta^{(l,k)}_{j,j}$	Per-component self-dots for s_{p_i} and s_j
$D^{\mathcal{L}}_{i,j}$	Accumulated cross dot over $\mathcal{L}$: $\sum_{(l,k)\in\mathcal{L}} \delta^{(l,k)}_{i,j}$
$\hat{\gamma}^{\mathcal{L}}_{i,j} = \dfrac{D^{\mathcal{L}}_{i,j}}{\sqrt{D^{\mathcal{L}}_{i,i}}\sqrt{D^{\mathcal{L}}_{j,j}}}$	Reconstructed cosine from selected components
$\hat{\Gamma}^{\mathcal{L}}$	Vector stacking all $\hat{\gamma}^{\mathcal{L}}_{i,j}$
$\rho(\mathcal{L}) = \mathrm{sim}_{\cos}\left(\hat{\Gamma}^{\mathcal{L}}, \Gamma^{\theta}\right)$	Agreement with full-model similarities

Table A.6.: Similarities, dot-product caches, and reconstructed scores.

Symbol	Meaning
$\mathrm{BM25}_D(s_{p_i}, s_j)$	BM25 score of s_j for query s_{p_i} over corpus D
$\mathcal{C}_i(b)$	Ordered index set of top-b BM25 candidates for s_{p_i} (contains i)
$\mathrm{accuracy}_\theta(D_p, D)$	Top-1 retrieval accuracy using full gradients over all j
$\mathrm{accuracy}^{(b)}_\theta(D_p, D)$	BM25-restricted accuracy (compare only $j \in \mathcal{C}_i(b)$)
$\mathrm{accuracy}^{(b)}_{\mathbf{W}^{(l,k)}}(D_p, D)$	Per-component accuracy (using $\mathbf{v}_{\mathbf{W}^{(l,k)}}$)
$\mathrm{accuracy}^{(b)}_{\mathcal{L}}(D_p, D)$	Accuracy using the selected set $\mathcal{L}$
ε	Small constant for numerical stability in divisions

Table A.7.: Candidate selection (BM25) and retrieval metrics.

Symbol	Meaning
$\mathcal{L}$	Set of selected layer components (grows during forward greedy)
$\mathcal{R} = \mathcal{W} \setminus \mathcal{L}$	Remaining components not yet selected
$((l_t, k_t),\, \rho_t)$	t-th selected component and running similarity score
$((l_t, k_t),\, a_t)$	t-th selected component and running accuracy score

Table A.8.: Greedy forward layer selection notation.

A.2. Preliminary Analysis of Gradient Dot Products

To validate the core assumption that paraphrased samples are most similar to their originals, a preliminary analysis was conducted on a small subset of the data. Figure A.1 visualizes the dot products of the full-model gradients between 25 paraphrased samples and their original counterparts. As illustrated there, the main diagonal contains significantly higher values, indicating that the model is able to find the corresponding original data points.

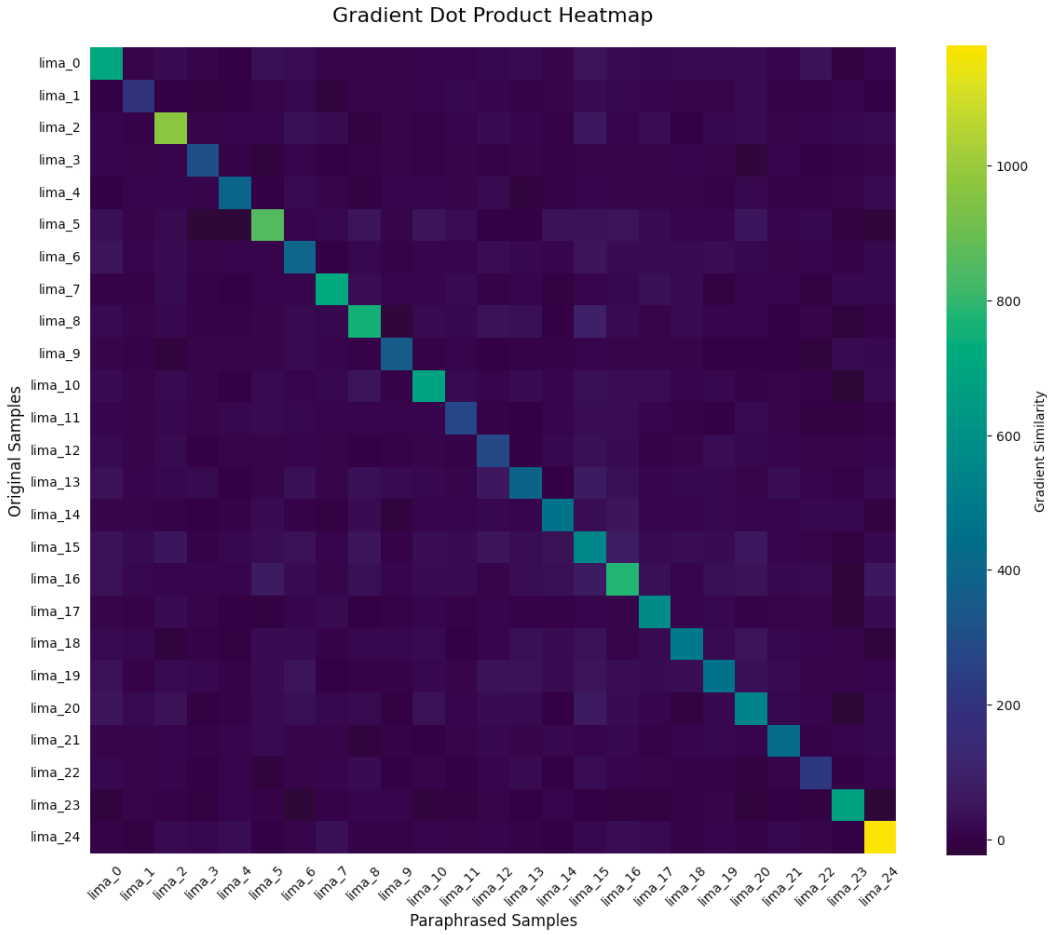


Figure A.1.: Preliminary analysis of gradient dot products between paraphrased and original samples.

A.3. Dataset Examples

This section provides concrete examples from the LIMA dataset to illustrate the transformations applied in the *paraphrased* and *model-generated* settings.

A.3.1. Paraphrased

The paraphrased equivalent for the presented example for the *LIMA* dataset illustrated in Subsection 4.1.2.

Paraphrased user message $s_{p_{0_P}}$:

```
Are brain cells capable of moving? Specifically, I'm referring to
long-distance migration, ideally occurring within the brain.
```

Paraphrased assistant message s_{p0_G}:

> The inquiry is quite extensive, and it's important to recognize that the brain is made up not only of neurons but also of glial cells (supporting cells) and pre-mitotic neuronal stem cells. Additionally, as critical colleagues in research have pointed out, the developmental stage is crucial, since the developing embryonic brain is significantly different from the fully developed adult brain. Nonetheless, after reviewing various studies, the answer to the question turns out to be surprisingly straightforward: Yes, brain cells do migrate. In the adult brain, glial cells are known to migrate (Klämbt, 2009). These glial cells perform numerous functions, with a prominent example being oligodendrocytes, which migrate over considerable distances to locate their target axons and encase them to create the insulating myelin sheath (Tsai and Miller, 2002). Neuronal stem cells also migrate long distances in response to injury (Imitola et al., 2004), moving from specific stem-cell areas (such as the hippocampus and subventricular zone) to other parts of the brain (Clarke, 2003). It has been demonstrated that post-mitotic but non-differentiated neurons can migrate within the adult brain in fish (Scott et al., 2012) and also in mammals and non-human primates (Sawada et al., 2011). Unsurprisingly, glial cells, stem cells, and neurons also undergo migration during embryonic development. Most notably, post-mitotic neurons that are intended for peripheral roles must migrate over relatively long distances from the neural crest to their designated target areas (Neuroscience, 2nd ed, Neuronal Migration).

A.3.2. Model-Generated

The model-generated equivalent for the presented example for the *LIMA* dataset illustrated in Subsection 4.1.2.

Model-generated user message $s_{p0_P} = s_{m0_P}$:

> Are brain cells capable of moving? Specifically, I'm referring to long-distance migration, ideally occurring within the brain.

Model-generated assistant message s_{m0_G}:

> Brain cells, such as neurons, are indeed capable of moving. This movement is crucial for various functions, including communication between neurons and the formation of neural networks. However, the extent of brain cell movement is limited by the physical constraints of the brain's structure.

```
Neurons are highly organized and interconnected, forming complex
networks that enable the brain to process and interpret information.
The movement of neurons within these networks is facilitated by the
cytoskeleton, which provides structural support and allows for the
movement of the cell body.

While brain cells can move within the brain, the movement is typically
limited to short distances and is not as extensive as the movement of
neurons in other parts of the body. The brain's structure and the
limitations of its physical constraints prevent the movement of brain
cells over long distances.
####
Yes, brain cells can move, but it is limited to short distances within
the brain.
```

A.4. Results

This section contains supplementary figures that provide a more detailed, layer-by-layer breakdown of the results presented in the main body of the thesis.

A.4.1. Accuracy per Layer Component

The following figure visualizes the retrieval accuracy for each individual layer component.

Paraphrased

Figure A.2 shows the retrieval accuracy for each layer component in the paraphrased setting, complementing the aggregated view in the main results chapter.

A.4.2. Comparison between Layer Components and Full Gradient

The following figures illustrate the cosine similarity between the score vectors of each individual layer component and the score vector of the full model gradient, broken down by layer depth.

Paraphrased

Figure A.3 details this comparison for the paraphrased setting.

Model-generated

Similarly, Figure A.4 shows the comparison for the model-generated setting.

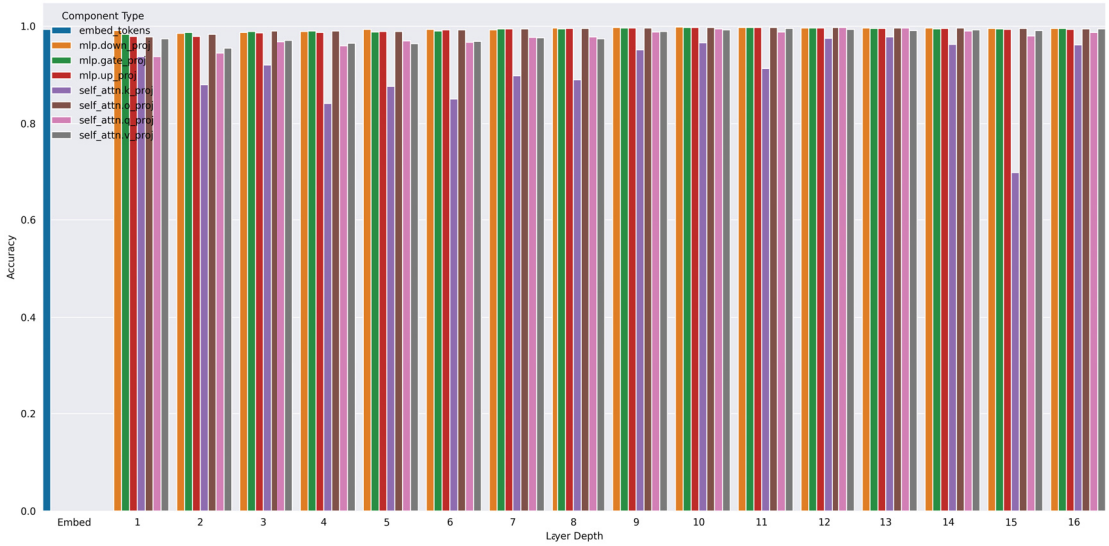


Figure A.2.: Accuracy per layer in the paraphrased setting

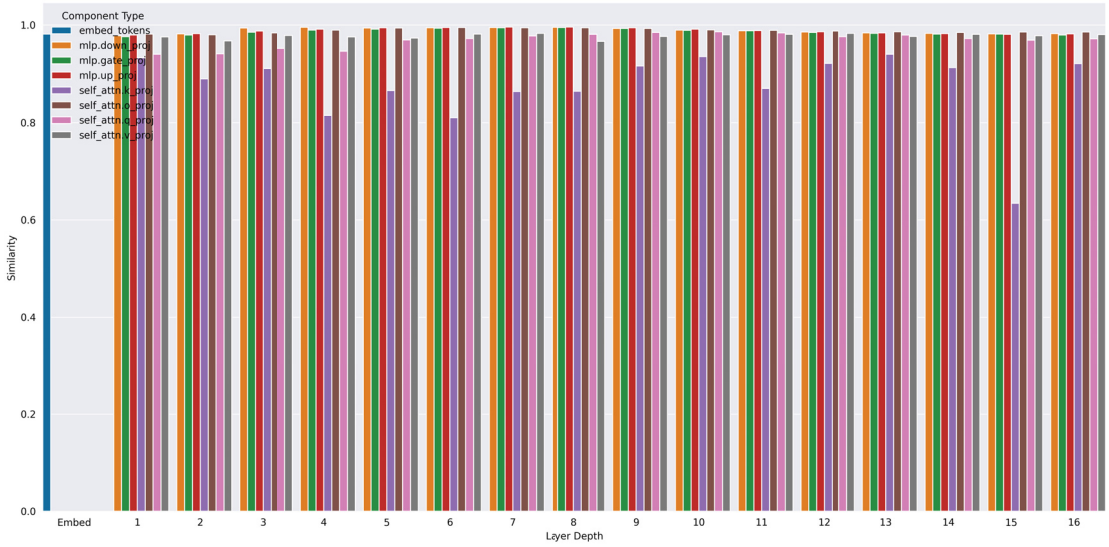


Figure A.3.: Comparison of single layer component and full model gradient scores for the paraphrased setting.

A.5. Code and Repository

The complete project code can be found on the following GitHub repository: `https://github.com/lukas-hinterleitner/master-thesis`.

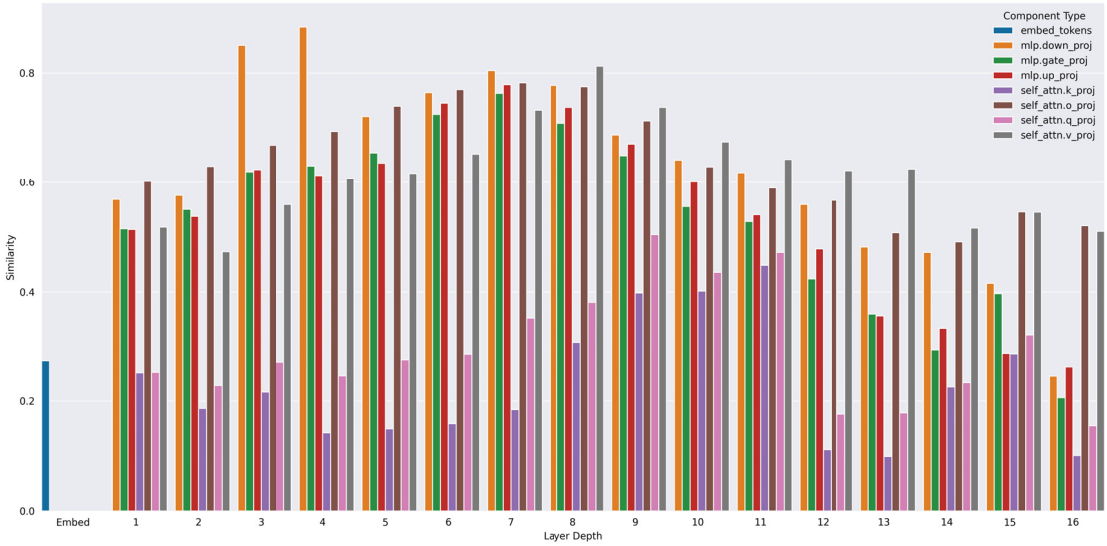


Figure A.4.: Comparison of single layer component and full model gradient scores for the model-generated setting.