POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Exploring Redundancy and Shared Representations for Transformer Models Optimization

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Enrico Simionato**

Student ID: 10698193
Advisor: Prof. Mark James Carman
Co-advisors: Vincenzo Scotti, Nicolò Brunello
Academic Year: 2023-24

Abstract

Large language models have revolutionized natural language processing enabling the deployment of advanced AI systems across a vast range of domains. Their rapid advances in language understanding and reasoning make them powerful tools, but their growing size leads to challenges in terms of efficiency, cost, and accessibility. Optimizing LLMs to minimize resource consumption is essential to ensure their sustainable use.

This thesis explores structural and weight redundancies in Transformer-based architectures, aiming to identify inefficiencies and leverage them through targeted compression techniques.

A central focus is assessing whether different modules perform overlapping functions. Although some degree of similarity is observed in the analyzed cases, redundancy proves to be lower than expected, challenging the assumption that weight matrices can be interchanged across layers without compromising performance. Additionally, an analysis of model matrices examines whether they exhibit an inherently low-rank structure.

To further explore these aspects, three novel compression methods are introduced: MASS, which enforces weight aggregation and sharing; GlobaL Fact, which factorizes matrices using shared components; and ABACO, which provides low-rank approximations through a continuous compression process. Experimental results indicate that while these approaches reduce the number of parameters, their limited ability to preserve performance hinders their practical viability.

The findings highlight the complexity of extracting redundancy from Transformer architectures, raising questions about its extent across layers and blocks. By addressing these challenges, this thesis aims to contribute to ongoing efforts to improve the efficiency of LLMs.

Keywords: natural language processing, large language models, redundancy, model compression, weight sharing, low-rank approximation.

Abstract in lingua italiana

I large language models hanno rivoluzionato l'elaborazione del linguaggio naturale, rendendo possibile l'implementazione di innovativi sistemi basati sull'intelligenza artificiale in un'ampia gamma di settori. Tuttavia la crescente complessità di queste architetture comporta sfide significative in termini di efficienza computazionale, costi e accessibilità. Ottimizzare tali modelli per ridurre il consumo di risorse è dunque essenziale per garantirne un utilizzo sostenibile.

Questa tesi indaga il livello di ridondanza presente nelle architetture basate sul Transformer, con l'obiettivo di identificare inefficienze e sfruttarle attraverso tecniche mirate di compressione.

L'analisi si focalizza sulla verifica di possibili sovrapposizioni funzionali tra moduli distinti. Sebbene nei casi analizzati emerga una minima similarità tra le loro operazioni, la ridondanza si rivela inferiore alle aspettative, mettendo in discussione l'ipotesi che le matrici dei pesi possano essere scambiate tra i layers senza deteriorare le prestazioni. Inoltre lo studio esamina la struttura delle matrici di diversi LLMs per verificare che esse presentino un rango ridotto.

Per sfruttare potenziali ridondanze, vengono introdotti tre nuovi metodi di compressione: MASS, che sfrutta l'aggregazione e la condivisione dei pesi; GlobaL Fact, che fattorizza i layers utilizzando matrici condivise; e ABACO, che fornisce approssimazioni a rango ridotto attraverso un processo di compressione continuo. I risultati sperimentali indicano che, sebbene questi approcci riducano il numero di parametri, la loro limitata capacità di preservare le prestazioni ne compromette le possibilità di utilizzo.

Le evidenze raccolte sottolineano la complessità insita nell'estrazione della ridondanza dai Transformers, sollevando interrogativi sulla sua effettiva estensione all'interno di layers e blocchi. Tramite le analisi condotte, questa tesi aspira a contribuire agli sforzi in corso per migliorare l'efficienza dei large language models.

Keywords: elaborazione del linguaggio naturale, modelli di linguaggio, ridondanza, compressione di modelli, condivisione dei pesi, approssimazione a rango ridotto.

Contents

List of Tables

List of Figures

List of Acronyms

ABACO	Adapter-Based Approximation and Compression Optimization
AdaLoRA	Adaptive Low-Rank Adaptation
AI	Artificial Intelligence
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
ASVD	Activation-aware Singular Value Decomposition
BERT	Bidirectional Encoder Representations from Transformers
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
DL	Deep Learning
ELU	Exponential Linear Unit
FFNN	Feed-Forward Neural Network
FWSVD	Fisher-Weighted Singular Value Decomposition
GAN	Generative Adversarial Network
GlobaL Fact	Global-Local Factorization
GQA	Grouped-Query Attention
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
GPT-3	Generative Pre-trained Transformer 3
GPT-4	Generative Pre-trained Transformer 4
GRU	Gated Recurrent Unit

Llama	Large Language Model Meta AI
LLM	Large Language Model
LoRA	Low-Rank Adaptation
LSTM	Long Short-Term Memory
MASS	Matrix Aggregation and Sharing Strategy
MHA	Multi-Head Attention
ML	Machine Learning
MLP	Multi-Layer Perceptron
MQA	Multi-Query Attention
SSE	Sum of Squared Errors
NER	Named Entity Recognition
NLP	Natural Language Processing
OBC	Optimal Brain Compression
OBS	Optimal Brain Surgeon
OCR	Optical Character Recognition
PCA	Principal Component Analysis
POS	Part-Of-Speech
PTQ	Post-Training Quantization
QAT	Quantization-Aware Training
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network
RoPE	Rotary Positional Encoding
RSSE	Relative Sum of Squared Errors
SGD	Stochastic Gradient Descent

SVD	Singular Value Decomposition
t-SNE	t-distributed Stochastic Neighbor Embedding
TPU	Tensor Processing Unit
TTS	Text-to-Speech
VeRA	Vector-based Random Matrix Adaptation

1 | Introduction

1.1. Introduction

Since the dawn of human history, speech has been the cornerstone of human interaction, fostering both individual cognitive development and the formation of complex societies. By enabling the exchange of ideas, knowledge, and emotions, it serves as the foundation of human life and progress. With the advent of writing, language evolved further, becoming an indispensable tool in every aspect of human life, from governance and commerce to art and science.

Over centuries, numerous disciplines, ranging from linguistics and philosophy to computer science, have sought to unravel the complexities of language. Over the past decades, the focus has shifted towards automating the comprehension, reasoning, and creation of language, revealing and enabling possibilities that were once only imaginable.

Advancements in technology have brought us to an era where enabling seamless communication between humans and computers is not just an aspiration but a rapidly evolving reality. In just a few years, the development of sophisticated mathematical models capable of writing, speaking, and reasoning has transformed artificial intelligence into a widely accessible tool. These innovations have extended their reach beyond academia, empowering non-expert users with intuitive and powerful capabilities.

Large Language Models (LLMs) stand at the forefront of this revolution, embodying decades of interdisciplinary research and innovation. By integrating advances in linguistics, mathematics, and computer science, LLMs harness vast datasets and sophisticated neural architectures to push the boundaries of machine comprehension and creativity. Their widespread adoption highlights their extraordinary potential to transform the way people interact with technology.

At their core, LLMs excel by capturing and predicting linguistic patterns. By distilling the intricacies of human expression into mathematical representations, they unlock unprecedented capabilities: generating coherent, contextually rich text; translating languages

with remarkable accuracy; summarizing information; and even engaging in nuanced conversations.

However, the immense power of these models comes at a significant cost. The tremendous size and computational demands of state-of-the-art LLMs present substantial challenges. Training large language models requires massive computational power, extensive memory, and considerable energy consumption. Even after training, the deployment phase can be resource-intensive. These requirements hinder the accessibility of such models, especially for organizations and researchers with limited resources. For now, a standard end-user machine cannot provide sufficient resources to train these models locally, and sometimes even running them is impossible without expensive hardware, such as powerful Graphics Processing Units (GPUs) to handle the heavy computations required.

Furthermore, the environmental impact of training these models has raised serious concerns, given the substantial carbon footprint associated with large-scale computation. The energy demands of training state-of-the-art LLMs highlight the urgent need for more efficient approaches.

In light of the challenges posed by the size and complexity of LLMs, recent research has focused on compression techniques to improve efficiency and accessibility. Methods such as quantization, pruning, and knowledge distillation have proven effective in reducing memory and computational demands while maintaining performance. Quantization optimizes resource usage by lowering the precision of model parameters. Pruning streamlines the model by removing redundant or less relevant components. Knowledge distillation replicates the functionality of a larger model within a smaller one by transferring its learned behavior. Among these methods, low-rank approximation, a mathematical approach that simplifies matrices by rewriting them with a product of smaller components, remains relatively underexplored. Its application to large language models is still in its early stages, making it an interesting area for further investigation and development.

1.2. Aims

This thesis aims to identify and exploit redundancies within and across the architectural components of Transformer-based models, targeting reductions in size and computational complexity without compromising their effectiveness. This work contributes to the ongoing efforts to make LLMs more accessible and deployable by minimizing their resource requirements.

Central to this investigation is the application of factorization-based compression meth-

ods, with a particular emphasis on low-rank approximations of model layers. These approaches systematically reduce unnecessary parameters and address redundancies in the large weight matrices characteristic of Transformers. By examining various forms and implementations of this technique, the thesis seeks to uncover practical methods for streamlining model components.

Ultimately, this research aspires to enhance the understanding of redundancy in Transformer-based LLMs and to propose novel compression techniques. By identifying redundancies both within and across components, this study explores methods to leverage internal and shared features and minimize unnecessary information. In particular, new methods are introduced to achieve a balance between efficiency and effectiveness:

- **Matrix Aggregation and Sharing Strategy (MASS)**;
- **Global-Local Factorization (GlobaL Fact)**;
- **Adapter-Based Approximation and Compression Optimization (ABACO)**.

The remarkable potential of large language models comes with significant challenges, including computational demands, environmental impact, and ethical concerns around equitable access and centralized control. These limitations risk excluding smaller organizations and researchers while raising questions about sustainability and social impact.

Compression techniques offer a solution by reducing resource requirements without compromising performance, democratizing access to LLMs and mitigating their environmental footprint. This thesis investigates innovative compression strategies, focusing on redundancies in Transformer architectures and methods like matrix factorization and adapter-based approaches, evaluating their effectiveness in balancing efficiency and performance.

By addressing these challenges, this research aims at contributing to the development of LLMs that are not only powerful but also equitable, sustainable, and aligned with ethical principles, fostering broader inclusion and responsible use of artificial intelligence.

1.3. Thesis Overview

This section provides an overview of the thesis structure and a brief summary of each chapter. This work is organized into eight chapters, each building on the previous one and contributing to a cohesive narrative that underpins the research objectives, key ideas, and findings.

Chapter 1: Introduction

The opening chapter sets the stage for the thesis, introducing the main topics and outlining the motivation and scope of the work.

Chapter 2: Background

This chapter delves into the core concepts of machine learning, deep learning, and natural language processing, with a specific emphasis on Transformer architectures, which are the primary focus of this research. It establishes the theoretical framework and technical grounding required to navigate the subsequent chapters.

Chapter 3: Related Works and State-of-the-Art Methods

A comprehensive review of the existing literature and state-of-the-art methods in model compression, with a particular focus on transformer model factorization, is presented in this chapter.

Chapter 4: Research Questions

This chapter reports the main research questions driving the thesis. It provides the rationale behind the chosen research direction, connecting theoretical inquiry with practical exploration.

Chapter 5: Metrics, Datasets, Models

This chapter presents the evaluation metrics, datasets, and models employed in this work. Together, these components form the backbone of a machine learning study.

Chapter 6: Compression Methods

The sixth chapter provides a detailed explanation of the proposed compression algorithms, thoroughly discussing their underlying rationale, advantages, and limitations.

Chapter 7: Experiments and Results

Experiments and results are presented in this chapter. A detailed analysis of Transformer models is conducted, with a focus on identifying redundancies. Furthermore, the proposed compression methods are evaluated through targeted experiments. The results are

analyzed and discussed to determine their effectiveness in preserving performance while enhancing resource efficiency.

Chapter 8: Conclusions and Future Developments

The final chapter reflects on the outcomes of the thesis, summarizing the key findings and their implications. It also identifies opportunities for future research, outlining potential extensions to further advance the field of deep model compression.

2 | Background

Artificial Intelligence (AI) is a multidisciplinary field of research focused on studying and developing methods, algorithms, and software that enable machines to perceive and interpret their environment, learn from experience and take actions to achieve specific goals. This field intersects and encompasses a wide range of fields, including computer vision, natural language processing, and robotics, aiming to create systems capable of autonomous decision-making and problem-solving on many different sources of information.

Natural Language Processing (NLP) is a subfield of artificial intelligence that focuses on the interaction between computers and human language. It involves developing algorithms and models that enable machines to understand, manipulate, and generate human language.

In the early stages of NLP, systems predominantly relied on rule-based approaches, where linguists and computer scientists manually developed comprehensive sets of linguistic rules to enable machines to process and understand human language. During the 1970s and 1980s, NLP research was centered on symbolic methods and expert systems, which encoded grammatical rules and knowledge in structured formats. These systems were rigid and lacked robustness, often failing to handle the ambiguities and complexities inherent in human language.

The late 1980s and 1990s marked a paradigm shift with the introduction of statistical methods and machine learning techniques in NLP. This transition was facilitated by two critical developments: the increasing availability of large text corpora and advances in computational power, which made data-driven approaches feasible.

Statistical methods enabled algorithms to learn linguistic patterns directly from data rather than relying on handcrafted rules. These approaches significantly improved the performance of NLP tasks and laid the groundwork for the modern era of NLP.

Over the past decade, deep learning has revolutionized natural language processing. Deep neural networks, especially those leveraging the Transformer architecture, have redefined performance standards across diverse NLP tasks. Today, tasks like machine translation,

sentiment analysis, and text generation are predominantly driven by advanced large-scale language models, including Generative Pre-trained Transformer 3 (GPT-3) [5], Generative Pre-trained Transformer 4 (GPT-4) [43], Mistral [28], Large Language Model Meta AI (Llama) 3 [12], and others.

Artificial intelligence, particularly in its intersection with natural language processing, represents a dynamic and rapidly evolving domain, characterized by relentless advancements that continually redefine the limits of innovation and practical application. A comprehensive review of the existing literature is essential to contextualize and guide meaningful research in this domain. This chapter aims to provide a detailed exploration of the foundational principles and recent progress in machine learning, deep learning, and NLP, setting the stage for the research presented in this thesis.

2.1. Machine Learning

Machine Learning (ML) is a subfield of artificial intelligence dedicated to creating algorithms and statistical models that enable systems to learn from data and make predictions or decisions. Unlike traditional programming, where explicit instructions are hardcoded for specific tasks, ML systems discover patterns and relationships in data, improving their performance over time with minimal human intervention.

Machine learning encompasses several key paradigms, each differing in the type of data used and the learning approach applied:

- **Supervised Learning**: Involves learning from labeled datasets, where the goal is to predict outputs for new, unseen inputs.
- **Unsupervised Learning**: Focuses on uncovering patterns, groupings, or structures within unlabeled data.
- **Reinforcement Learning**: Involves training an agent, acting in a given environment, to make optimal decisions in order to achieve a specific goal.

In addition to these main paradigms, there are also variations that build upon them:

- **Semi-Supervised Learning**: Combines small amounts of labeled data with large quantities of unlabeled data to improve learning efficiency.
- **Self-Supervised Learning**: A special case of unsupervised learning where pseudo-labels are automatically generated from the data itself, enabling the model to learn meaningful representations without manual annotations.

The primary focus of this thesis is on the supervised learning and self-supervised paradigms and their application within the field of natural language processing.

2.1.1. Supervised Learning

Supervised learning aims to learn a mapping function from inputs to outputs to predict the output for new, unseen inputs. Given a dataset of examples $\mathcal{D} = < x_i, t_i >$, where x_i represents the input features and t_i the corresponding target, he objective is to learn the function f such that $t = f(x)$.

The nature of the target t determines the type of supervised learning problem:

1. If the target is a discrete value, the problem is categorized as a classification problem. For example, determining whether an email is spam or not.
2. If the target is a continuous value, the problem is classified as a regression problem. An example would be predicting house prices based on features like location and size.
3. If the target represents a probability, the problem is considered a probability estimation problem, such as estimating the likelihood of rain given weather conditions.

Supervised learning makes sense it is adopted and is practically used in context where the mapping between inputs and outputs and the of the setting is unknown and no mathematical rule or formula of the dynamics is know yet. Even if the dynamics is known but too complex to model or compute, machine learning can be an appropriate approach to provide results in a less resource expensive manner.

Supervised learning is particularly applicable in contexts where the relationship between inputs and outputs is unknown or too complex to model explicitly using mathematical rules or formulas. Even when a mathematical representation of the dynamics is available, it may be computationally expensive or impractical to use. In such cases, machine learning offers an efficient alternative to achieve accurate predictions with less computational overhead.

Supervised learning is particularly applicable in contexts where the relationship between inputs and outputs is unknown or too complex to model explicitly using mathematical rules or formulas. Even when a mathematical representation of the dynamics is available, it may be computationally expensive or impractical to use. In such cases, machine learning offers an efficient alternative to achieve accurate predictions with less computational overhead.

Learning a model

A supervised machine learning model is a mathematical function trained to map input data to output predictions. The goal of the learning process is to find the function that best approximates the underlying phenomenon.

The standard pipeline for solving a problem using machine learning is:

1. **Data Collection**: Gathering a substantial volume of relevant data is a fundamental preliminary step in any machine learning process. Both the quality and quantity of the data play a pivotal role in determining the performance of the resulting model.
2. **Data Preparation**: Preprocessing the collected data to ensure it is in a suitable format for computation. This step may include cleaning the data, handling missing values, normalization, and transforming raw inputs into meaningful features. In traditional machine learning, feature extraction, identifying and selecting the most informative attributes, is often crucial. However, in modern approaches, models automatically learn features directly from raw data, bypassing manual feature engineering. This is achieved through deep learning.
3. **Model Choice**: Choosing an appropriate model architecture or algorithm based on the nature of the problem, data characteristics, and computational constraints. The choice of model plays a significant role in balancing complexity, performance, and efficiency.
4. **Model Training**: Training the selected model involves optimizing its parameters using the training dataset. An optimization method, such as gradient descent, is used to minimize the error between the model's predictions and the actual targets. The objective is to approximate the target function as closely as possible.
5. **Hyperparameter Tuning**: Adjusting hyperparameters, which are variables that control the learning process but are not updated during training, can help refine the model. These hyperparameters are often tuned using a validation set to enhance the training process and improve performance.
6. **Evaluation**: Assessing the model's generalization ability on unseen data using a test set. A separate test dataset ensures an unbiased evaluation of the model's performance and helps detect issues like overfitting.
7. **Prediction**: Deploying the trained model to make predictions on new, unseen inputs. This step translates the learned relationships into practical, real-world outcomes.

While this pipeline provides a general framework, variations exist depending on the problem context.

The pipeline outlined above provides a structured approach to solving problems using machine learning, guiding each stage from data collection to prediction. Once a task and its corresponding dataset are defined within the supervised learning framework, three fundamental elements must be carefully specified to effectively tackle the training process:

- **Hypothesis Space**: The hypothesis space defines the set of all possible functions the machine learning model can represent. These functions serve as approximations of the target relationship between inputs and outputs. If the true target function lies within this space, the learning process has the potential to converge to an accurate solution. Otherwise, a bias will inevitably be introduced, limiting the model's performance.
- **Loss Function**: The loss function measures the discrepancy between the model's predictions and the actual target values. By quantifying this distance, the loss function provides a clear objective for the training process to minimize errors and improve accuracy.
- **Optimization Method**: The optimization method is the algorithm responsible for adjusting the model's parameters to minimize the loss function, thereby bringing the model's predictions as close as possible to the target values. Common examples of optimization methods include closed-form solutions, gradient descent, and Newton's methods.

Bias-Variance Trade-Off

An essential concept to consider when selecting the hypothesis space and training a model is the bias-variance trade-off. Bias-variance trade-off is a fundamental concept in machine learning and statistics that describes the relationship between a model's complexity, its prediction accuracy, and its ability to generalize to unseen data.

The total prediction error made by a model can be expressed as the sum of bias, variance, and an irreducible noise, which comes from the uncertainty in the data. Importantly, bias and variance are inversely related to model complexity: increasing the complexity of a model reduces bias but increases variance, while simplifying the model reduces variance but increases bias.

- **Bias** refers to the error introduced by approximating a target function with a simpler model. High bias occurs when a model is too simplistic and fails to capture the

underlying patterns in the data, resulting in a situation with high approximation error called **underfitting**.

- **Variance** measures how much the model's predictions fluctuate when trained on different datasets derived from the same data distribution. High variance occurs when the model is overly complex, capturing noise in the training data, which leads to **overfitting**. The model is memorizing the training examples instead of generalizing to unseen data. One effective way to reduce variance is to increase the amount of available training data.

Achieving an optimal balance between bias and variance is critical for building a model that generalizes well. This balance also depends on the complexity of the problem, the volume and quality of the available data, and the computational resources at hand.

Selecting an appropriate model complexity based on these considerations ensures that the machine learning process delivers effective and reliable results.

Parametric vs. Non-Parametric Models

- **Parametric Models**: These models assume a fixed number of parameters, regardless of the dataset size. Once the model structure is defined, its complexity remains constant. Examples include linear regression and logistic regression. Parametric models are computationally efficient but may struggle with complex relationships if the assumptions about the data are incorrect.

- **Non-Parametric Models**: These models adapt their complexity to the data, allowing the effective number of parameters to grow with the dataset size. Examples include decision trees, k-nearest neighbors, and support vector machines. While non-parametric models are more flexible, they require larger datasets and higher computational resources.

Frequentist vs. Bayesian Approaches

- **Frequentist Methods**: These approaches estimate model parameters by maximizing the likelihood of the observed data. Frequentist inference assumes fixed but unknown parameters and does not incorporate prior knowledge about them.

- **Bayesian Methods**: These approaches incorporate prior beliefs about the parameters in the form of a probability distribution and update these beliefs as new data is observed. Bayesian methods provide a probabilistic interpretation of parameter estimates and are particularly useful when uncertainty estimation is important.

Direct, Probabilistic, and Generative Models

- **Direct Models**: These models directly map inputs to outputs. An example is linear regression, where the relationship between input features and outputs is explicitly modeled.
- **Probabilistic Models**: These models estimate the probability distribution of outcomes given the input data. For instance, logistic regression estimates the likelihood of a binary outcome.
- **Generative Models**: These models learn the joint probability distribution of inputs and outputs. By modeling the data distribution, generative models can generate new samples. Examples include naive Bayes classifiers and Generative Adversarial Networks (GANs).

2.1.2. Unsupervised Learning

In unsupervised learning, the objective is to discover meaningful patterns, structures, or relationships within data that has no explicit labels. Given an unlabeled dataset of examples $\mathcal{D} = < x_i >$, the learning process focuses on revealing the inherent relationships in the data.

Key techniques in unsupervised learning include:

- **Clustering**: Partitioning the data into groups, or clusters, based on similarity. A widely used method is k-means clustering, which minimizes intra-cluster distances to identify natural groupings. Applications include customer segmentation, anomaly detection, and biological data analysis.
- **Dimensionality Reduction**: Reducing the number of input features while preserving the essential structure of the data. Techniques such as Principal Component Analysis (PCA) and t-distributed Stochastic Neighbor Embedding (t-SNE))project high-dimensional data into lower-dimensional spaces for visualization, noise reduction, and feature extraction.

Unsupervised learning is particularly valuable for exploratory data analysis, enabling insights into large and complex datasets where no prior labels exist. It also serves as a foundation for semi-supervised and self-supervised learning approaches.

2.1.3. Semi-supervised learning

Semi-supervised learning is a hybrid approach that lies between supervised and unsupervised learning. It uses a small amount of labeled data alongside a large amount of unlabeled data to improve learning performance. The key idea is that labeled data is often expensive and time-consuming to obtain, while unlabeled data is abundant and easy to collect.

Semi-supervised learning leverages the structure or patterns in the unlabeled data to augment and guide the training of the model. The labeled examples provide supervision, while the unlabeled data helps to regularize the model, reducing overfitting and improving generalization.

2.1.4. Self-supervised learning

Self-supervised learning is an emerging paradigm that learns representations from unlabeled data by creating pseudo-labels automatically from the data itself. It is often considered a form of unsupervised learning but with a pretext task that generates supervisory signals.

Self-supervised learning defines a pre-text task, a task designed to generate labels from the data's intrinsic structure. The model is first trained to solve this pretext task, and the learned representations are then transferred to downstream tasks, such as classification or regression.

Self-supervised learning eliminates the need for manually labeled data, as it leverages the inherent structure of the input data. It is particularly useful when labeled data are scarce, but large-scale unlabeled data are available.

Examples of pretext tasks include predicting the rotation angle of an image, solving jigsaw puzzles, or inpainting missing parts of an image in the domain of computer vision. In natural language processing, pretext tasks often involve predicting missing words in a sentence, as seen in models like Bidirectional Encoder Representations from Transformers (BERT) [10], or generating the next token in a sequence, as demonstrated by Generative Pre-trained Transformer (GPT) [44][45].

2.1.5. Reinforcement Learning

Reinforcement Learning (RL) operates in a setting where an agent performs sequences of actions within a given environment. The goal of RL is to determine the optimal actions

that the agent must take to achieve a predefined objective. This objective is formalized through a reward function, which evaluates the quality of each action performed by the agent and guides its learning process.

The foundation of RL lies in the Reward Hypothesis, as stated by Richard Sutton: *All of what we mean by goals and purposes can be well thought of as the maximization of the expected value of the cumulative sum of a received scalar signal (called reward).*

In this framework, the agent interacts with the environment over discrete time steps. At each step t:

- The agent observes the state of the environment s_t.
- It selects an action a_t to perform based on its current policy.
- The environment responds by transitioning to a new state (s_{t+1}) and providing a reward r_{t+1} that quantifies the immediate outcome of the action.

These interactions can be represented as sequences of experience tuples:

$$\{ < s_t, a_t, s_{t+1}, r_{t+1} >\}.$$

The goal of RL is to find an optimal policy, which is a mapping from states to actions, that maximizes the agent's performance measure. This measure is often defined as the discounted cumulative reward over time:

$$R = \sum_{t=0}^{T} \gamma^t r_{t+1},$$

where $\gamma \in [0, 1]$ is the discount factor that determines the relative importance of immediate versus future rewards, and T represents the time horizon, which can either be finite or infinite.

The agent improves its policy through trial and error, iteratively exploring and evaluating the outcomes of its actions. A central challenge in RL is balancing exploration, which refers to trying new actions to discover better strategies, and exploitation, which means leveraging known actions to maximize immediate rewards.

Reinforcement learning excels at solving sequential decision-making problems, where the consequences of actions unfold over time. Its ability to model adaptive, goal-oriented behavior has enabled significant advancements in fields such as game-playing systems (e.g., AlphaGo), autonomous robotics, and self-driving vehicles. By formalizing objec-

tives through the reward function, RL provides a robust framework for learning complex strategies in uncertain and dynamic environments.

2.2. Deep Learning

Deep Learning (DL) is a specialized subset of machine learning that focuses on using artificial neural networks as models to solve complex tasks. Its defining characteristic lies in the ability to automatically learn data features during training, which sets it apart from traditional machine learning techniques.

Unlike conventional machine learning approaches that depend on manually engineered features, deep learning allows models to discover the most relevant feature representations directly from the data, particularly when the data is unstructured. This capability enables deep neural networks to serve as powerful, general-purpose feature extractors, adapting to the specific requirements of the task at hand.

A key advantage of deep learning is its ability to facilitate end-to-end learning, where models can perform a task directly from raw inputs without requiring intermediate feature extraction steps. For example, in image recognition, a deep learning model can learn to identify objects directly from pixel values, bypassing the need for handcrafted visual descriptors.

The term "deep" in deep learning refers to the depth of the neural networks, meaning that these models consist of multiple layers of interconnected neurons. The depth of the network allows for the extraction of hierarchical and high-level abstractions, making it possible to learn complex patterns and relationships in data.

Deep learning has revolutionized numerous fields, including computer vision, speech recognition, and natural language processing. Its success is largely attributed to its ability to process vast amounts of data and extract meaningful representations that enable breakthroughs in solving tasks previously considered intractable.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the biological brain. They consist of interconnected nodes, called neurons, which are organized into groups called layers. Each neuron is connected to others through weighted links, and its activation depends on the signals it receives from connected neurons. In their simplest form, neurons activate when the combined signal from their inputs surpasses a certain threshold.

Different types of networks exist, varying in how neurons are interconnected and the internal operations performed. While ANNs are general-purpose models, certain architectures are more efficient for specific tasks and contexts.

Structure of a Neural Network

The neuron is the fundamental computational unit of a neural network. Each neuron receives a set of inputs, processes them through a weighted sum, adds a bias, and applies an activation function to produce its output.

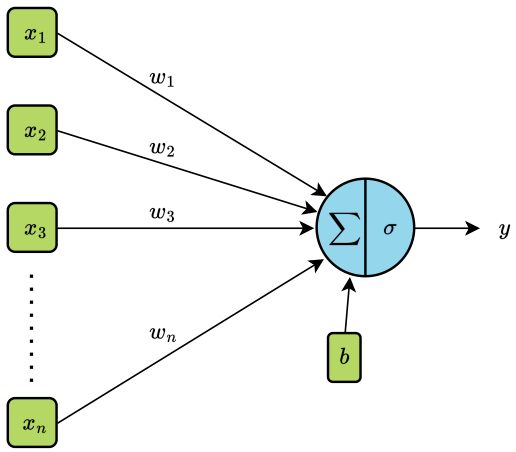


Figure 2.1: Illustration of a neuron in an artificial neural network.

Mathematically, a neuron computes its output y as follows:

$$y = \sigma\left(\mathbf{x} \cdot \mathbf{w}^T + b\right) = \sigma\left(\sum_{i=0}^{n} x_i \cdot w_i + b\right)$$

where:

- $\mathbf{w} \in \mathbb{R}^n$ is the weight vector,
- b is the bias,
- $\mathbf{x} \in \mathbb{R}^n$ represents the input vector having components $\langle x_1, x_2, ..., x_n \rangle$, and
- σ is the activation function.

In this thesis, vectors will be represented as row vectors unless otherwise specified. The same theory can equivalently be expressed using column vectors, with minor adjustments

to the notation.

A Feed-Forward Neural Network (FFNN), or Multi-Layer Perceptron (MLP), is the simplest neural network architecture. It consists of a sequence of layers with an arbitrary number of neurons per layer. Each neuron receives input from all neurons in the previous layer and passes its output to all neurons in the next layer. Information flows in a single direction, progressing sequentially through the network without loops, which would reintroduce outputs back into earlier computations, or shortcuts that bypass intermediate stages.

The structure of an FFNN can be divided into three types of layers:

- Input Layer: The input layer consists of the raw input features of the dataset, with each neuron corresponding to one feature dimension. It serves as the interface between the data and the network.
- Hidden Layers: Intermediate layers where inputs are transformed through multiple stages of processing. The depth and size of these layers determine the network's capacity to learn complex patterns.
- Output Layer: The final layer that produces the network's predictions. Its size matches the dimensions of the output.

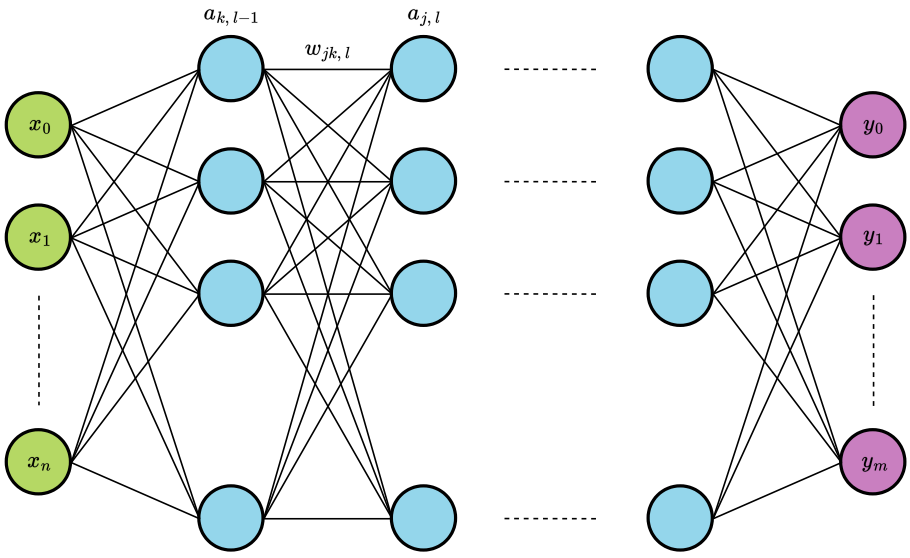


Figure 2.2: Illustration of the structure of a feed forward neural network. The input layer is shown in green, the hidden layers are depicted in light blue, and the output layer is highlighted in purple.

Operations within a single layer l, producing activation $\mathbf{a}_l = \langle a_{1,l}, a_{2,l}, ..., a_{n_l,l} \rangle$, can be formalized in vector notation as:

$$\mathbf{a}_l = \sigma\left(\mathbf{a}_{l-1} \cdot \mathbf{W}_l^T + \mathbf{b}_l\right),$$

and in scalar notation as:

$$a_{j,l} = \sigma\left(\sum_{k=0}^{n_{l-1}-1} a_{k,l-1} \cdot w_{jk,l} + b_{j,l}\right),$$

where:

- $\mathbf{W}_l \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix connecting layer $l-1$ to layer l,
- $\mathbf{b}_l \in \mathbb{R}^{n_l}$ is the bias vector for layer l,
- $\mathbf{a}_{l-1} \in \mathbb{R}^{n_{l-1}}$ is the activation vector of the previous layer,
- $w_{jk,l}$ is the weight connecting neuron k in layer $l-1$ to neuron j in layer l,
- $b_{j,l}$ is the bias of neuron j in layer l,
- $a_{k,l-1}$ is the activation of neuron k in the previous layer $l-1$,
- σ is the activation function, applied element-wise in vector notation or individually in scalar notation, and
- n_{l-1} and n_l represent the number of neurons in layer $l-1$ and l, respectively.

Traditional activation functions, such as Rectified Linear Unit (ReLU), Sigmoid, and Tanh, introduce non-linearity into neural networks, enabling them to model complex relationships in data. More advanced activation functions, such as Leaky ReLU, Exponential Linear Unit (ELU), and Swish, have been developed to address challenges like small gradients and slow convergence in artificial neural networks.

Training a Neural Network

Training a neural network involves learning the optimal weights and biases to minimize a loss function, which measures the difference between the network's predictions and the actual target values. The training process uses backpropagation in conjunction with optimization algorithms to iteratively update the model parameters.

The training loop can be divided into four main steps:

1. **Forward Propagation**: The input data passes through the network, layer by layer, to produce an output.
2. **Loss Calculation**: The network output is compared with the target values using a loss function, which quantifies the error.
3. **Backpropagation**: The gradients of the loss with respect to the network weights and biases are computed using the chain rule.
4. **Weight Update**: Weights and biases are adjusted using an optimization algorithm, such as Stochastic Gradient Descent (SGD) or its variants, to minimize the loss.

This process is repeated iteratively over multiple passes through the training data until the model converges to an optimal set of parameters.

A significant theoretical property of artificial neural networks is established by the universal approximation theorem, which states that a feed forward neural network with at least one hidden layer and a non-linear activation function can approximate any continuous function to an arbitrary degree of accuracy, given sufficient neurons in the hidden layer. This result highlights the power of neural networks as universal function approximators, enabling them to tackle a wide range of complex tasks.

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) [18] [33] are specialized neural network architectures designed to process and analyze image data. From a data perspective, an RGB image is represented as a three-dimensional tensor, where each cell corresponds to the intensity of a specific color channel for a given pixel. A crucial aspect of image data is that not only are the channel values significant, but the spatial arrangement of these values is equally important. The interactions between neighboring pixels convey meaningful patterns, enabling the extraction of high-level features, similar to how humans perceive visual elements.

CNNs are specifically engineered to leverage the spatial structure of images. Their architecture extracts meaningful information and detects patterns essential for solving a given task. During training, CNNs function as feature extractors, learning which features are most relevant for accomplishing the task at hand.

The input to a CNN undergoes a series of transformations, progressively generating a compressed representation of the image known as the latent representation. This representation captures the semantic meaning of the input image by encoding its essential visual features and is subsequently used to produce the final output of the network.

The architecture of CNNs is composed of several types of layers, each contributing to the overall functionality of the network:

1. **Convolutional Layers**: Convolutional layers are defined by three-dimensional filters, also known as kernels, which are learnable parameters of the layer. These filters are convolved with the input activations, which are raw images in the first layer and feature maps in deeper layers. Kernels identify patterns that the network learns during the training process to be relevant for solving the task. As the network progresses through successive convolutional layers, it detects increasingly complex and abstract patterns. Early layers activate features corresponding to low-level details such as edges and textures, while deeper layers respond to higher-level features, such as shapes or semantic components of the input image.

   The translation invariance property of convolution enables CNNs to detect patterns regardless of their spatial location within an image, making them robust to positional shifts.

2. **Non-Linear Activation**: Non-linear activation functions are applied after convolutions to introduce non-linearity into the model. Since convolutions are linear operations, these activations are necessary to enable the network to model complex, non-linear functions.

3. **Pooling Layers**: Pooling layers, such as max pooling or average pooling, reduce the spatial dimensions of feature maps while retaining the most important information. By summarizing features, these layers help the network gain a more global understanding of the input and increase the receptive field, which is the region of the input image influencing a given activation.

4. **Fully Connected Layers**: Fully connected layers are typically placed at the end of the network. They take the latent representation generated by the feature extraction layers and map it to the desired output space. These layers are responsible for tasks such as classification, where high-level information from the input is used to produce the final prediction.

5. **Transpose Convolutional Layers**: Transpose convolutional layers perform the reverse operation of standard convolutional layers, effectively upsampling the input activations. These layers expand the spatial dimensions of the input by inserting zeros between elements before applying the convolution operation. Transpose convolutions are commonly used in tasks like image segmentation and image reconstruction, where fine-grained visual details need to be generated from a latent

representation.

Convolutional neural networks have achieved remarkable success in various computer vision applications. In image classification, CNNs excel at identifying and categorizing objects within an image. They are also highly effective in object detection, where the goal is to locate and classify objects in an image. For image segmentation, CNNs are employed to divide an image into meaningful regions or segments, enabling detailed analysis and understanding of its content. Additionally, CNNs are widely used in image generation tasks, creating realistic images, as demonstrated in applications like GANs.

2.3. Sequence Data Problems

Sequence modeling problems involve data represented as ordered sequences, where the arrangement of elements holds essential meaning. The goal of these problems is to learn patterns, dependencies, or relationships within the sequences to perform specific tasks. Such problems often arise in domains where the temporal or sequential relationships between elements are critical.

These problems are classified based on the relationship between the number of inputs and outputs, reflecting the diverse nature of tasks encountered in various domains.

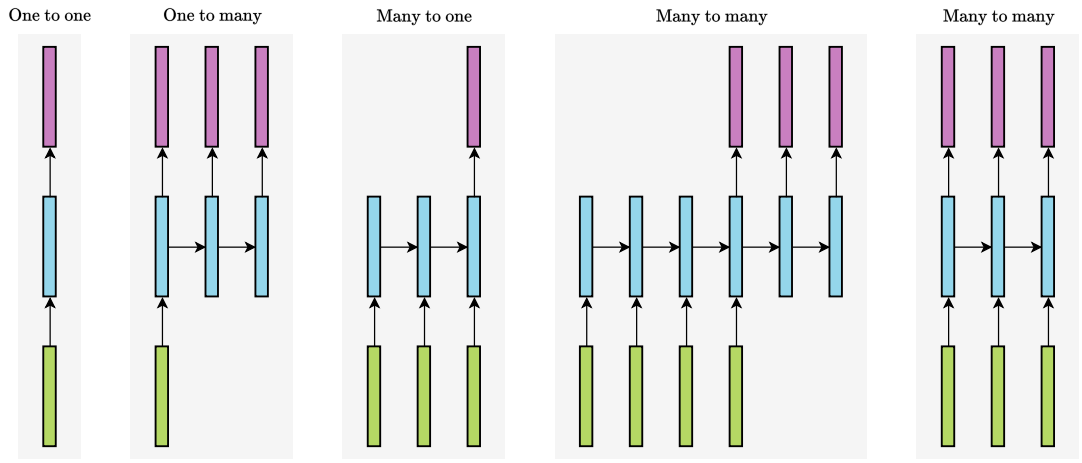


Figure 2.3: Classification of sequence modeling tasks based on input-output relationships.

One-to-One (Single Input, Single Output)

One-to-one represents the simplest case of sequence modeling, where a fixed-sized input is directly mapped to a fixed-sized single output. In this scenario, the sequence aspect is degenerate, as the data is treated as an isolated instance without sequential relationships.

For instance, in image classification, each image (input) is independently assigned a single category label (output).

One-to-Many (Single Input, Sequence of Outputs)

In a one-to-many setting, the input has a fixed size, while the output consists of a sequence with variable length. An input is used to generates a sequence of related elements. An example is image captioning, where one image produces a descriptive sentence.

Many-to-One (Sequence of Inputs, Single Output)

In this configuration, a sequence of inputs is transformed into a single output, a structure commonly employed in sequence classification tasks. For example, sentiment analysis evaluates a sequence of words within a sentence to determine a single sentiment label, such as positive or negative.

Many-to-Many (Sequence of Inputs, Sequence of Outputs)

In this configuration, a sequence of inputs is mapped to a sequence of outputs of the same length, where each input element corresponds directly to a specific output element. This structure is commonly used in tasks that require a one-to-one mapping between elements of the input and output sequences.

A classic example is part-of-speech tagging, where each word in a sentence is assigned its grammatical role. Another application is video frame classification, where each frame in a sequence of video frames is labeled with a corresponding category or annotation.

Many-to-Many with Different Lengths (Sequence of Inputs, Sequence of Outputs)

In this more general setup, a sequence of inputs is mapped to a sequence of outputs, but the lengths of the two sequences can differ. This configuration is particularly useful for tasks where the input and output sequences represent different types of information or have varying granularity.

Examples include machine translation, where a sentence in one language is translated into its equivalent in another language, and text summarization, where a lengthy input document is condensed into a shorter version that retains its essential meaning and content. This structure is also used in applications like speech-to-text conversion, where an audio waveform (input sequence) is transcribed into a sequence of words (output sequence).

Among sequential modeling problems, natural language processing, which deals primarily with sequences of words, has gained remarkable prominence in recent years. It has emerged as one of the most dynamic and influential areas of modern sequence modeling research.

2.4. Natural Language Processing

Natural language processing is a multidisciplinary field at the intersection of computer science, artificial intelligence, linguistics, and philosophy. NLP employs computational techniques to analyze, interpret, and generate natural language in a way that is meaningful and contextually relevant. Today, it focuses on enabling machines to process, understand, and generate human language, both written and spoken. NLP is increasingly integrated into everyday life, assisting people in solving a wide range of problems and enhancing human-computer interaction. By bridging the gap between human language and machine processing, NLP facilitates tasks such as understanding commands, generating coherent text, and retrieving critical information.

Challenges in Natural Language Processing

The complexity of NLP stems from the intrinsic characteristics of human language. Languages are highly expressive, capable of conveying abstract concepts, emotions, and even nonsensical ideas. Sentences often carry ambiguity or imprecision, making their interpretation context-dependent.

Another practical challenge is the diversity of languages, each with unique grammatical and structural nuances. Some languages have rich linguistic resources, while others lack extensive corpora, making it harder to train effective models across all languages.

Despite these obstacles, NLP has evolved significantly over the years. The field has shifted from rule-based systems and statistical methods to modern deep learning approaches, which now deliver state-of-the-art performance across a wide array of tasks. This transformation has been driven by advances in computational power, the availability of large-scale textual datasets, and the development of cutting-edge architectures.

2.4.1. Tasks in Natural Language Processing

NLP encompasses a variety of tasks aimed at facilitating human-computer interaction and extracting meaningful information from language. These include:

- **Text-only tasks**:

- **Discriminative tasks**:
  - **Text classification**: Assigning predefined categories to input text documents. Applications include spam detection in emails and sentiment analysis, where textual content is analyzed to determine its emotional tone or polarity.
  - **Part-Of-Speech (POS) Tagging**: Identifying and labeling the grammatical categories (e.g., noun, verb, adjective) of each word within a sentence.
  - **Named Entity Recognition (NER)**: Detecting and classifying named entities within a text, such as persons, organizations, locations, and dates.
  - **Co-reference Resolution**: Resolving pronouns and other references within a text to enhance coherence and facilitate understanding.
  - **Anonymization**: Removing or masking identifying information from textual data to protect privacy and ensure that individuals or entities mentioned in documents remain unrecognizable.
- **Information Retrieval and Extraction**:
  1. **Information Retrieval**: Identifying and ranking relevant content from extensive data corpora based on user queries.
  2. **Information Extraction**: Automatically extracting structured information from unstructured text, facilitating knowledge discovery and data organization.
- **Generative tasks**:
  - **Machine Translation**: : Automatically translating text from one language to another.
  - **Text Summarization**: Generating concise and coherent summaries from longer textual content while preserving essential information.
  - **Paraphrase Generation**: Rewriting text to preserve its original meaning while improving fluency or adapting it for different contexts.
  - **Question Answering**: Designing systems capable of interpreting natural language queries and providing precise, context-aware answers.
  - **Language Generation**: Producing fluent, coherent, and contextually

relevant text based on a given input, with applications in creative writing, news generation, and code synthesis.

    * **Human-Machine Interaction**: Enabling AI-driven systems, such as virtual assistants and chatbots, to generate and respond to human input in a coherent, contextually appropriate and natural way, fostering seamless interaction.

- **Multimodal tasks**:
  - **Speech-related tasks**:
    * **Automatic Speech Recognition (ASR)**: Converting spoken language into written text.
    * **Text-to-Speech (TTS)**: Generating spoken audio from textual input.
  - **Image-related tasks**:
    * **Optical Character Recognition (OCR)**: Converting scanned images of text into digital text.
    * **Image Captioning**: Writing a caption to the image given as input.

Many of these tasks, such as part-of-speech tagging, are nearly solved, while others, considered more challenging, continue to see significant advancements as NLP models improve.

List 2.4.1 provides a non-exhaustive overview of NLP tasks, as natural language processing methods are highly versatile and continuously evolving, becoming more powerful and adaptable to new challenges every day.

The successful resolution of these tasks brings numerous benefits:

- **Accessibility**: Facilitating seamless communication between humans and machines, enabling users to interact with technology through natural language.
- **Efficiency**: Automating the retrieval, processing, and analysis of large volumes of text data, saving time and resources for individuals and organizations.
- **Insight Extraction**: Providing meaningful insights from unstructured data, such as social media posts, reviews, and articles, to support decision-making and trend analysis.
- **Global Reach**: breaking language barriers by providing translation and multilingual support, fostering global communication.

Word Embeddings

Many machine learning approaches, particularly deep learning methods, require numerical input data. However, language is inherently symbolic, consisting of elements such as characters or words, which must be transformed into numerical representations for computational processing.

A straightforward method for representing words in a vocabulary is one-hot encoding, where each word is encoded as a vector the same length as the vocabulary. This vector is filled with zeros, except for the index corresponding to the word, which is set to one. While simple, this representation is not semantically meaningful. It assigns equal distance between all word pairs, regardless of their actual semantic or contextual relationships.

Word embeddings offer a more sophisticated and powerful alternative for mapping words to numerical vectors. Word embeddings transform high-dimensional discrete input spaces into continuous lower-dimensional vector spaces. This transformation leverages deep learning models to produce dense vector representations that encode semantic properties. In these embeddings, spatial relationships between vectors reflect the semantic relationships between words. For instance, similar words are placed closer together in the vector space, and linear transformations capture meaningful analogies. As an example, the vector difference between "king" and "queen" mirrors the one between "man" and "woman."

Word2Vec

Word2Vec [40], developed by Google, is a foundational algorithm to generate word embeddings of words that encode their semantic and contextual relationships. It employs two primary architectures: the Continuous Bag of Words (CBOW) model and the Skip-Gram model. Both models rely on a shallow neural network with a distinctive structure. The input and output layers contain neurons corresponding to the vocabulary size, while the hidden layer acts as a bottleneck with fewer neurons, creating a compressed representation of the data.

During training, the network adjusts the weights of the hidden layer to minimize the prediction error for an auxiliary task, which leverages the relationships between words and their context. The two architectures differ in their approach:

1. The CBOW model predicts a target word based on its surrounding context, synthesizing information from neighboring words.

2. The Skip-Gram model, on the other hand, predicts the surrounding context given a target word and is particularly effective at learning embeddings for infrequent

words.

The bottleneck structure of the hidden layer ensures that words are compressed into a dense, lower-dimensional space. This compressed representation is rich enough to reconstruct a word or its context, encapsulating the semantic and contextual nuances of each term. Once training concludes, the optimized weights of the hidden layer form a matrix that maps words to their corresponding vectors. These vectors constitute the learned word embeddings.

The introduction of word embeddings has significantly enhanced the performance of nearly all machine learning methods in natural language processing. Their ability to provide richer and more meaningful semantic representations has advanced a wide array of NLP tasks, from text classification to machine translation.

Recurrent Neural Networks

Recurrent Neural Networks (RNNs) [14] [29] are artificial neural networks specifically designed to process sequential data, such as time-series data, natural language, and speech. Solving such problems often requires the ability to remember past inputs and understand the dependencies between elements in a sequence.

RNNs address this need by incorporating a memory mechanism known as the hidden state, which retains information about previous inputs to influence the current output. This memory is implemented through recurrent connections that feed the network's output back into its input, enabling it to model dependencies across time steps. Essentially, the hidden state evolves with each time step, acting as a distributed representation of the sequence's history.

While recurrent neural networks effectively model dependencies within sequences, they face significant challenges when handling long-term dependencies due to the vanishing gradient problem. This issue arises when gradients diminish as they propagate backward through deep networks or recurrent time steps. Small derivatives from activation functions like sigmoid and tanh, combined with repeated multiplications, cause the gradients to shrink exponentially. As a result, earlier layers or time steps receive minimal updates, impeding the network's ability to learn from distant events.

To mitigate these challenges, researchers have introduced several strategies. These include employing activation functions less prone to vanishing gradients (e.g., ReLU), using more sophisticated weight initialization techniques, and developing specialized architectures like long short-term memory networks and gated recurrent units.

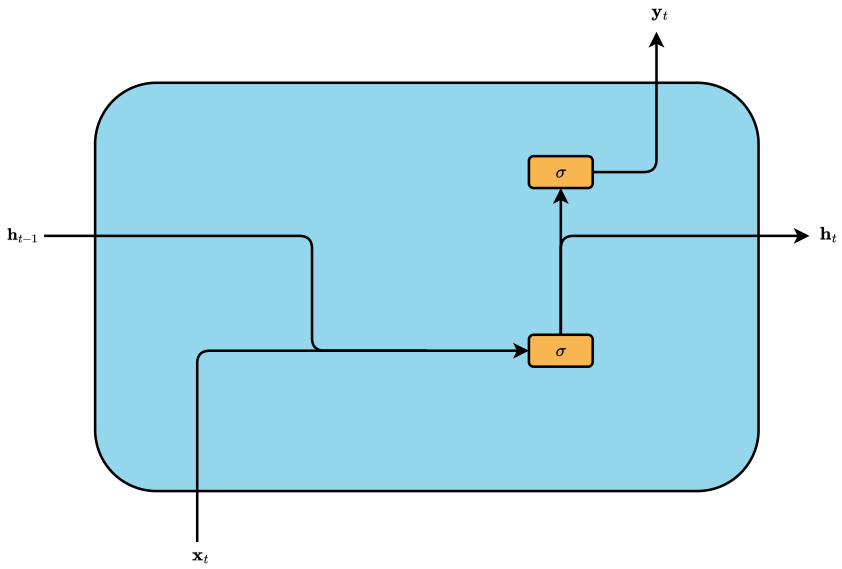


Figure 2.4: Illustration of a recurrent neural network.

Long-Short Term Memory

Long Short-Term Memorys (LSTMs) [25] are a refined form of RNNs designed specifically to address the vanishing gradient problem. By integrating a dedicated cell state and a set of gating mechanisms, LSTMs can capture long-range dependencies in sequential data more effectively than traditional RNNs. Their architecture allows them to selectively preserve or discard information, enabling them to retain valuable context over extended sequences.

LSTM contains three types of gates that enable different behaviors:

1. **Memory Gate or Cell State**: Acting as the long-term memory of the LSTM, the cell state carries information across multiple time steps, remaining largely unchanged unless explicitly modified by other gates. This design ensures that important details can be preserved over extended periods.

2. **Forget Gate**: This gate determines which information should be discarded from the cell state. By processing the current input x_t and the previous hidden state h_{t-1} through a sigmoid function, it produces values between 0 and 1 to scale the cell state contents, effectively filtering out irrelevant details.

3. **Input Gate**: The input gate decides what information should be added to the cell state by processing the current input x_t and the previous hidden state h_{t-1}. The gate uses a hyperbolic tangent function to generate a candidate signal representing

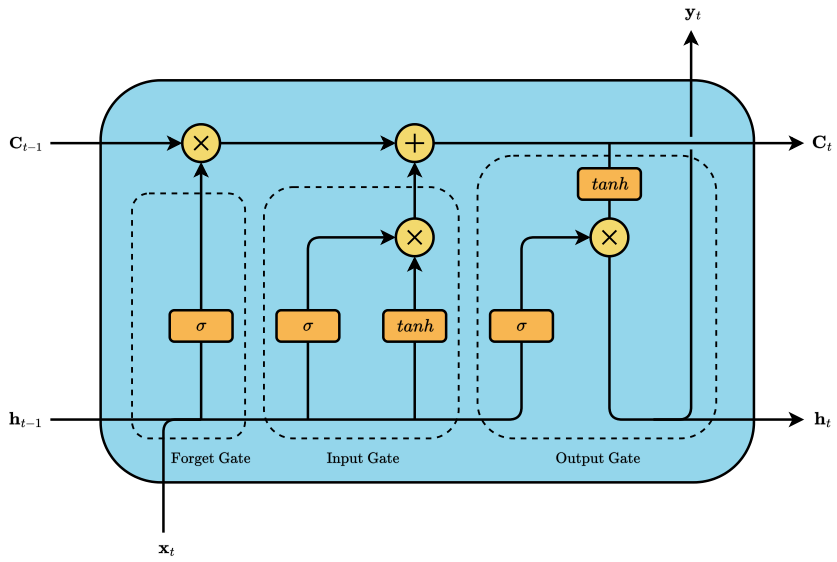


Figure 2.5: Illustration of a long short-term memory unit.

potential new information. Simultaneously, a sigmoid function calculates the degree to which this candidate information should influence the cell state. By combining these two outputs, the input gate ensures that only relevant and appropriately scaled information is integrated into the cell state, enabling the model to update its memory.

4. **Output Gate**: The output gate controls how much of the information in the cell state is passed to the hidden state h_t, which serves as the output of the LSTM at each time step. To achieve this, the gate processes the current input x_t and the previous hidden state h_{t-1} through a sigmoid function, producing a scaling factor. Simultaneously, the cell state undergoes a transformation using a hyperbolic tangent function to regulate the information being revealed. The final hidden state is then calculated by scaling the transformed cell state with the output gate's signal, ensuring that only the most relevant information is propagated.

Gated Recurrent Unit

A variant of the LSTM, the Gated Recurrent Unit (GRU) [8], simplifies the architecture by combining the forget and input gates into a single update gate. Additionally, GRUs integrate the hidden and cell states into a unified representation, reducing computational overhead while retaining effectiveness for many sequential tasks.

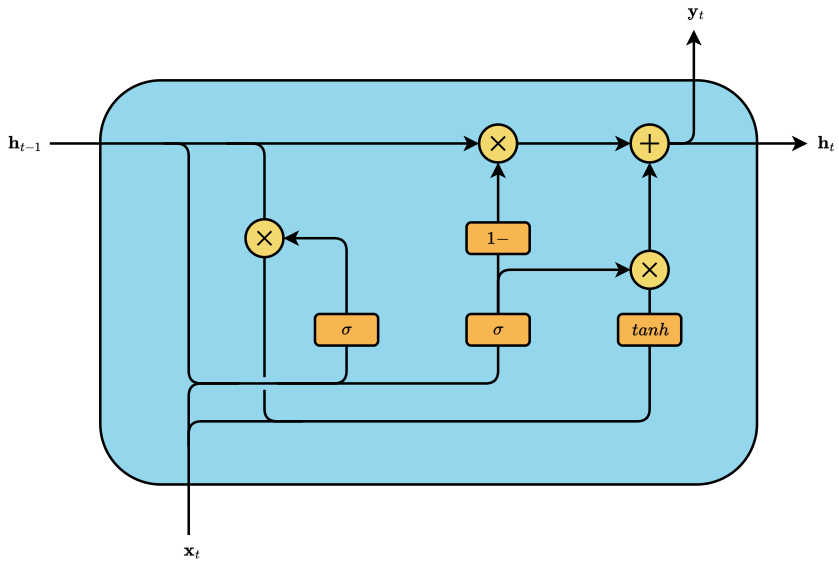


Figure 2.6: Illustration of a gated recurrent unit.

Leveraging their cell state and gating mechanisms, LSTMs and GRUs excel at capturing long-term dependencies and context in sequential data. This capability makes them particularly robust for tasks that demand an understanding of relationships spanning extended time periods.

Encoder-Decoder Models

Encoder-decoder models provide a powerful framework for tackling sequence-to-sequence tasks. They operate by processing an input sequence and distilling its essential information into a compact representation, which then guides the generation of an appropriate output sequence.

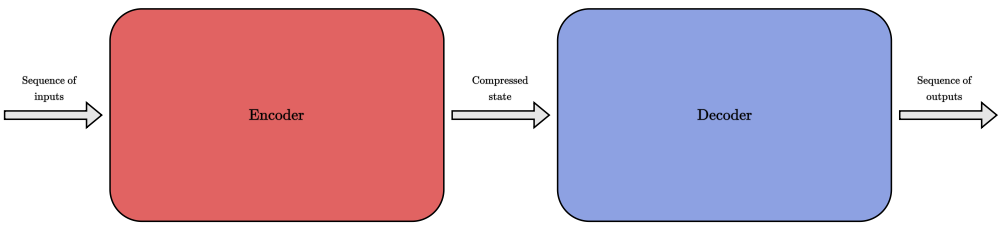


Figure 2.7: Schematic illustration of an encoder-decoder architecture.

These architectures comprise two core components: an encoder, which reads and encodes the input sequence into a fixed-size context vector (or latent representation), and a de-

coder, which uses this compressed representation to produce the corresponding output sequence.

Encoder Stage The encoder processes the input sequence token by token, gradually updating its hidden state to incorporate information from the current token as well as all preceding ones. In early designs, LSTM-based or GRU-based encoders were common, ensuring that the final hidden state captured all crucial details of the input. For example, in a machine translation scenario, the encoder would read a source-language sentence and encode it into a context vector intended to represent the meaning of the entire sentence.

Decoder Stage The decoder is responsible for generating the output sequence. It produces the sequence one token at a time, guided by its current hidden state, which is initialized using the encoder's final hidden state. The context vector from the encoder provides a representation of the input, which is then dynamically refined by the decoder as it processes the generated output tokens, shaping its hidden state at each time step. For instance, in a machine translation scenario, the decoder begins with the encoded representation of the input sentence and generates the translated sentence one word at a time.

A major drawback of encoder-decoder models based on recurrent neural networks is their reliance on compressing the entire input sequence into a single context vector. From the encoder's perspective, effectively encoding the input sequence into a fixed-length vector can be challenging, especially for long and complex inputs. Similarly, the decoder might require different pieces of information from the input sequence at various stages of its computation, which the fixed-size representation may fail to provide. This extreme compression creates a representation bottleneck that significantly hinders performance, particularly for long sequences. Additionally, the one-at-a-time encoding and generation approach is inherently slow, as each token depends on the complete processing of its predecessors. This sequential processing limits the model's efficiency and scalability.

Attention Mechanism

Attention mechanisms were introduced to address the representation bottleneck in encoder-decoder models. Instead of relying solely on a single fixed-size context vector, attention enables the decoder to dynamically focus on different parts of the input sequence at each generation step. This approach provides richer, more detailed information, overcoming the limitations of compact hidden representations.

First proposed by Bahdanau et al. [4], attention mechanisms transformed sequence-to-sequence modeling by allowing the decoder to selectively attend to the most relevant

segments of the input sequence during each decoding step. At each decoding step, the decoder computes a context vector as a weighted sum of the encoder's hidden states. The weights, determined by a learnable scoring function, indicate the relevance of each input token to the current decoding step. The context vector, combined with the decoder's hidden state, guides the prediction of the next token, enabling more accurate and context-aware generation. This innovation significantly improved tasks like machine translation by alleviating the constraints imposed by fixed-size representations.

While attention mechanisms effectively solve the representation bottleneck, they still rely on sequential processing, where each step depends on the completion of the previous one. This limitation was addressed by self-attention mechanisms, which process input sequences in parallel and serve as the foundation for the Transformer architecture.

2.4.2. Transformers

The Transformer model, introduced by Vaswani et al. in 2017 in the landmark paper *Attention is All You Need* [53], marked a significant turning point in the field of natural language processing. This architecture has become the foundation of many modern advancements in NLP, driving significant performance improvements across nearly all tasks.

The Transformer introduced two groundbreaking innovations: the self-attention mechanism and the positional encoding which are discussed in the following subsections.

Transformers are highly scalable, with the ability to increase layers and parameters to achieve better performance. This scalability has been instrumental in the emergence of large language models such as GPT, Llama, and Mistral, which have achieved state-of-the-art results in various NLP applications.

In its original formulation, the architecture consists of two primary components: an encoder and a decoder, each composed of several identical blocks stacked sequentially. Each block integrates two key components: a self-attention mechanism and a feed forward neural network. These elements are complemented by residual connections and layer normalization, which enhance the stability of the model and facilitate efficient gradient flow during training.

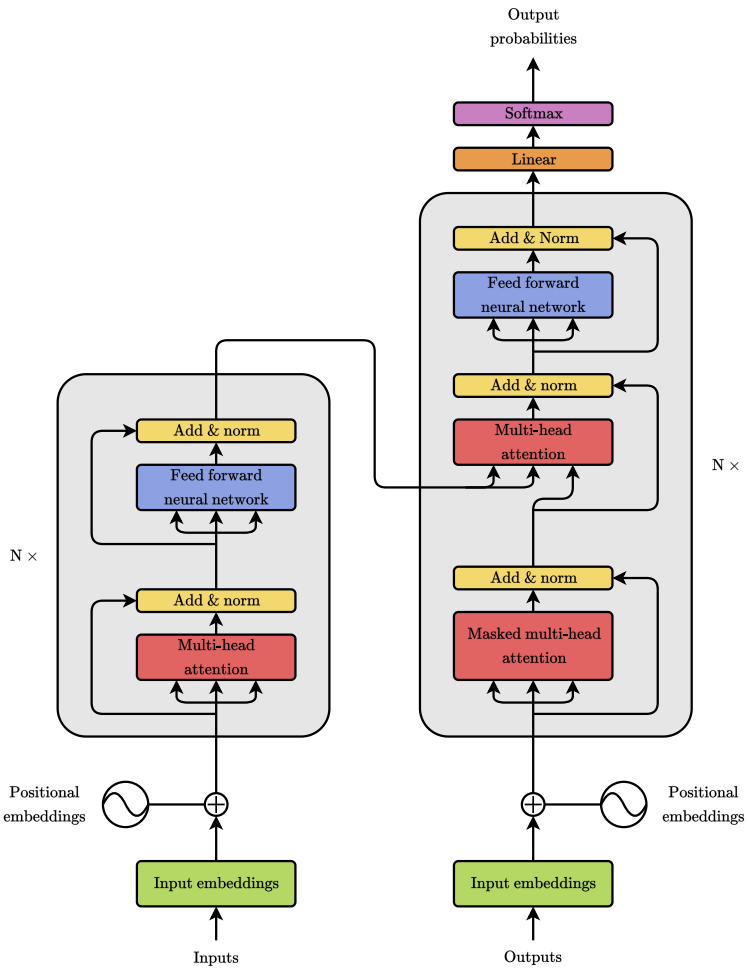
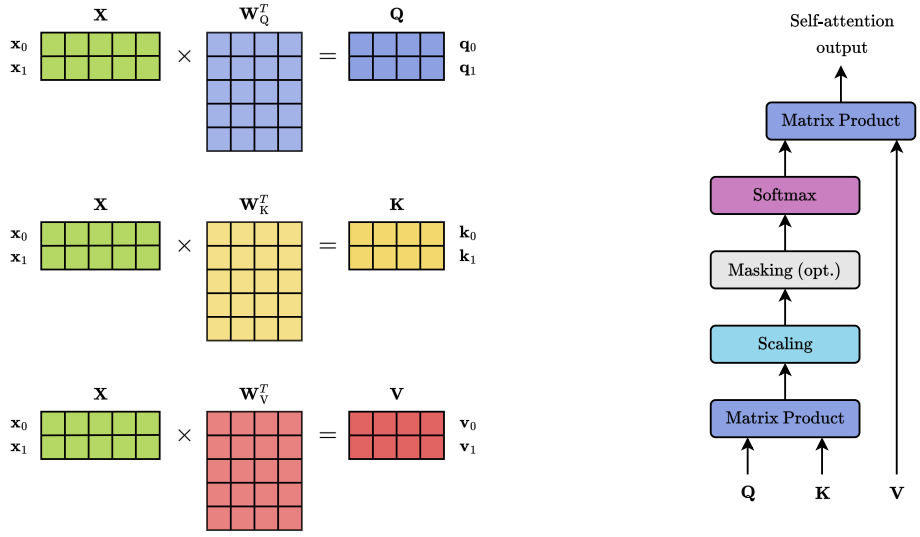


Figure 2.8: Illustration of the original Transformer model from the paper *Attention is All You Need*.

Self-Attention Mechanism

The self-attention [1] is a computational module designed to evaluate the relationships between tokens within a sequence, dynamically weighing their relevance at each step. By calculating attention scores for each token relative to every other token, self-attention effectively captures contextual and long-range dependencies. Unlike earlier architectures such as LSTM-based encoder-decoder models, which process inputs sequentially, this mechanism leverages parallel processing of tokens. This significantly enhances computational efficiency while enabling the model to derive a comprehensive and context-aware understanding of the input. This capability is fundamental to solving a wide range of tasks in natural language processing.

The input to the self-attention mechanism is a sequence of token embeddings. For each



(a) Computation of the query, key and value matrices form the self-attention weights and input tokens.

(b) Illustration of the operations performed by the self-attention module.

Figure 2.9: Illustrations of the self-attention mechanism

token in the sequence, self-attention generates three vectors through linear transformations, each serving a specific role. These transformations are defined by weight matrices that are learned during the training process:

1. The query vector $\mathbf{q}$ encodes what the current token is looking for in other tokens. The aggregation of these vectors across all tokens forms the query matrix ($\mathbf{Q}$).
2. The key vector $\mathbf{k}$ represents the token's own content and features to be matched by other tokens. The key vectors for all tokens are combined to construct the key matrix ($\mathbf{K}$).
3. The value vector $\mathbf{v}$ holds the actual information about the token that will be integrated into the output. The value vectors of different inputs combine to form the value matrix ($\mathbf{V}$).

Given an embedding matrix $\mathbf{X} \in \mathbb{R}^{l \times d_h}$, where l is the sequence length and d_h is the embedding dimension, the query $\mathbf{q}_i$, key $\mathbf{k}_i$, and value $\mathbf{v}_i$ of the i-th token are computed as:

$$\mathbf{q}_i = \mathbf{x}_i \cdot \mathbf{W}_{\mathrm{Q}}^T, \quad \mathbf{k}_i = \mathbf{x}_i \cdot \mathbf{W}_{\mathrm{K}}^T, \quad \mathbf{v} = \mathbf{x}_i \cdot \mathbf{W}_{\mathrm{V}}^T.$$

For the entire sequence, these vectors are combined into matrices $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$, expressed as:

$$\mathbf{Q} = \mathbf{X} \cdot \mathbf{W}_{\mathrm{Q}}^{T}, \quad \mathbf{K} = \mathbf{X} \cdot \mathbf{W}_{\mathrm{K}}^{T}, \quad \mathbf{V} = \mathbf{X} \cdot \mathbf{W}_{\mathrm{V}}^{T},$$

where $\mathbf{W}_{\mathrm{Q}}$, $\mathbf{W}_{\mathrm{K}}$, and $\mathbf{W}_{\mathrm{V}} \in \mathbb{R}^{d_h \times d_h}$ are learnable weight matrices refined during training.

Once the query and key vectors are obtained, a similarity measure is computed for each pair of tokens by taking the dot product between the query vector of one token and the key vector of another. To stabilize gradients and prevent excessively large values, the result is scaled by dividing it by $\sqrt{d_k}$, where d_k is the dimensionality of the key vector. These scaled values are then passed through a softmax function, which normalizes them so that the attention scores for each token sum to 1. These coefficients represent the "focus" or "importance" that each token assigns to others in the sequence.

The attention scores a_{ij} between tokens i and j is defined as:

$$a_{ij} = \mathrm{softmax}\left(\frac{\mathbf{Q}_i \cdot \mathbf{K}_j^{T}}{\sqrt{d_k}}\right)$$

In matrix form, the operation is expressed as:

$$\mathbf{A} = \mathrm{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^{T}}{\sqrt{d_k}}\right).$$

The attention scores are subsequently applied to compute a weighted sum of the value vectors. For token i, the self-attention output is:

$$\textbf{Self-attention}\left(\mathbf{x}_i \mid \mathbf{X}, \mathbf{W}_{\mathrm{Q}}, \mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}}\right)_i = \mathbf{y}_i = \sum_j \alpha_{ij} \mathbf{V}_j.$$

In matrix notation, this can be compactly represented as:

$$\textbf{Self-attention}\left(\mathbf{X} \mid \mathbf{W}_{\mathrm{Q}}, \mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}}\right) = \mathbf{Y} = \mathbf{A} \cdot \mathbf{V} = \mathrm{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^{T}}{\sqrt{d_k}}\right) \cdot \mathbf{V}.$$

This computation produces a new representation for each token, influenced by the entire sequence, with greater emphasis on tokens assigned higher attention weights.

The self-attention mechanism empowers Transformers to effectively model both local and global dependencies. By allowing each token to interact with all others, Transformers excel at capturing intricate patterns in sequences, making them highly versatile for a range of applications.

Multi-Head Attention

Multi-Head Attention (MHA) is the standard approach used in Transformer models to enhance the representational power of the attention mechanism. It extends the self-attention process by introducing multiple "heads", each learning unique attention patterns. This design enables the model to capture a diverse range of relationships and dependencies within the input sequence.

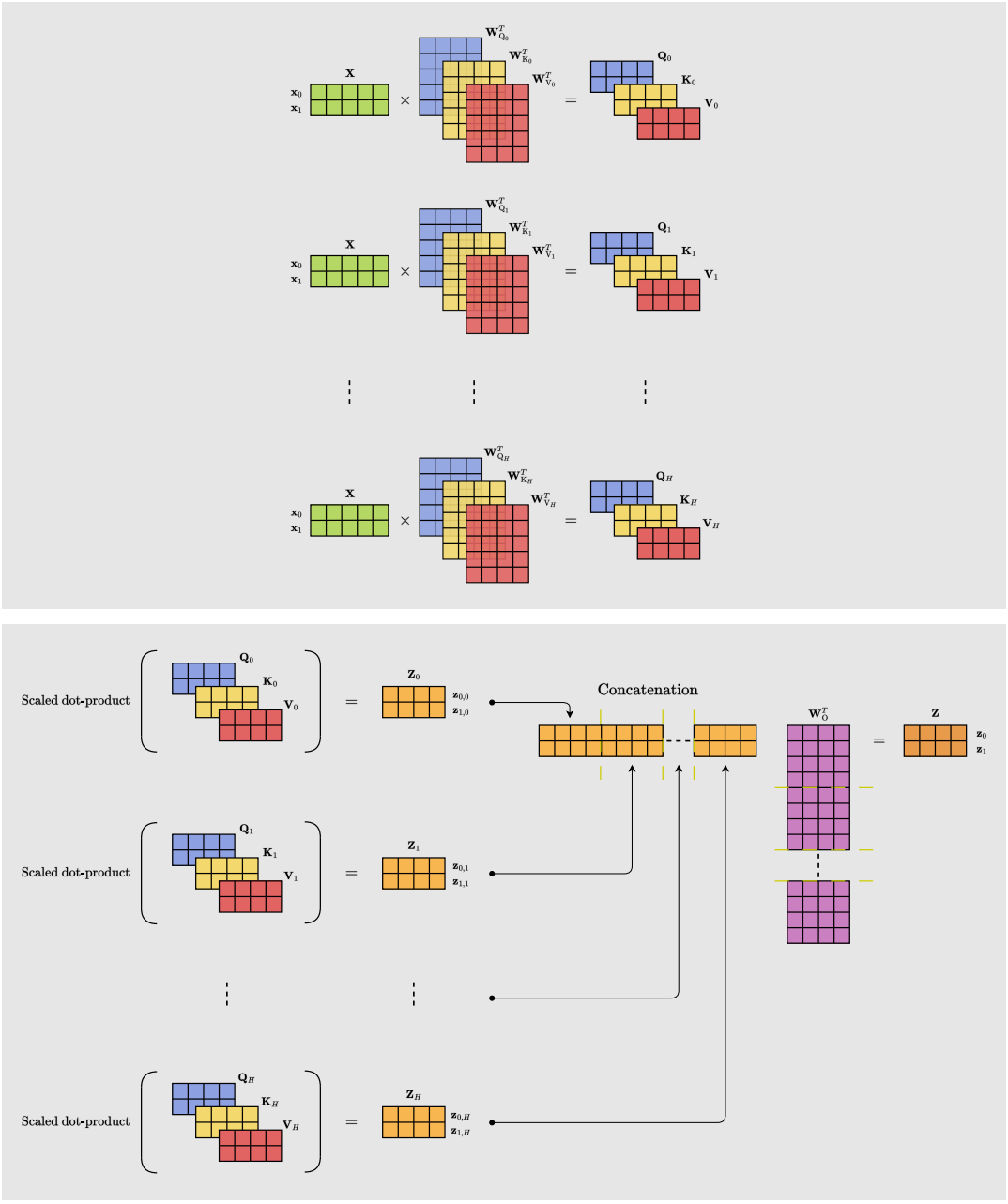


Figure 2.10: Schema of the multi-head self-attention computation.

In this mechanism, the input embeddings are projected into separate query, key, and value spaces for each head. Each head independently computes attention, yielding h distinct outputs. These outputs are then concatenated and passed through a final linear layer to generate the combined result. This multi-headed structure allows the model to simultaneously focus on different aspects of the input.

Figure 2.11: Illustration of how query, key, and value heads are associated in multi-head self-attention. For each position of the input sequence $\{i, i+1, \dots\}$, H independent sets of query, key and value heads are computed. In particular, the j-th heads operates on the queries $\{\mathbf{q}_{i,j}, \mathbf{q}_{i+1,j}, \dots\}$, keys $\{\mathbf{k}_{i,j}, \mathbf{k}_{i+1,j}, \dots\}$, and values $\{\mathbf{v}_{i,j}, \mathbf{v}_{i+1,j}, \dots\}$.

Each head computes its own set of attention weights and outputs, allowing the model to capture different types of relationships between tokens. The input embeddings are projected into multiple query, key, and value spaces and each head performs self-attention independently, using a unique projections, resulting in H different outputs. The outputs from all heads are concatenated and passed through a final linear transformation to produce the overall result.

The computation for multi-head attention can be expressed as:

$$\textbf{Multi-head}\left(\mathbf{X} \mid \mathbf{W}_{\mathrm{Q}}, \mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}}\right) = \mathrm{Concat}\left(\textbf{head}_1, \textbf{head}_2, \dots, \textbf{head}_H\right) \cdot \mathbf{W_O}^T,$$

where each head is calculated independently as:

$$\mathbf{head}_i = \mathbf{Self\text{-}attention}\left(\mathbf{X} \mid \mathbf{W}_{\mathrm{Q}_i}, \mathbf{W}_{\mathrm{K}_i}, \mathbf{W}_{\mathrm{V}_i}\right).$$

Here:

- $\mathbf{W}_{\mathbf{Q}_i}$, $\mathbf{W}_{\mathbf{K}_i}$ and $\mathbf{W}_{\mathbf{V}_i}$ are the learned weight matrices for the i-th head.
- $\mathbf{W}_{\mathrm{O}}$ is the learned output projection matrix applied after concatenating the head outputs.

For computational efficiency, these individual weight matrices are often represented as concatenated matrices, here denoted as $\mathbf{W}_{\mathrm{Q}}, \mathbf{W}_{\mathrm{K}}, \mathbf{W}_{\mathrm{V}}$.

Multi-Query Attention

Multi-Query Attention (MQA) streamlines the multi-head attention mechanism by employing a single set of keys and values shared across all heads, while preserving distinct queries for each head. This design allows each head to uniquely interact with the shared keys and values, enabling the model to focus on diverse aspects of the input sequence.

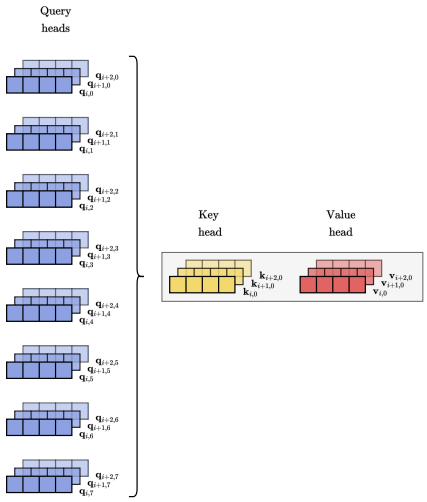


Figure 2.12: Illustration of how query, key, and value heads are associated in multi-query attention. For each position of the input sequence $\{i, i+1, \dots\}$, H independent sets of query heads are computed. However, all heads share the same key and value vectors. In particular, the j-th head operates on the queries $\{\mathbf{q}_{i,j}, \mathbf{q}_{i+1,j}, \dots\}$, while all heads use the shared keys $\{\mathbf{k}_{i,0}, \mathbf{k}_{i+1,0}, \dots\}$ and values $\{\mathbf{v}_{i,0}, \mathbf{v}_{i+1,0}, \dots\}$.

Empirical studies demonstrate that this approach significantly reduces memory and computational overhead while maintaining comparable performance to standard multi-head

attention. Its efficiency makes it especially well-suited for decoder layers and tasks demanding fast, resource-efficient calculation.

Grouped-Query Attention

Grouped-Query Attention (GQA) strikes a balance between multi-head and multi-query attention. In this approach, the attention heads are partitioned into distinct groups, with each group sharing a common set of keys and values. This architecture effectively reduces memory consumption relative to traditional multi-head attention while retaining diversity across groups. By offering a compromise between representational richness and computational efficiency, grouped-query attention provides a versatile solution for a wide range of applications.

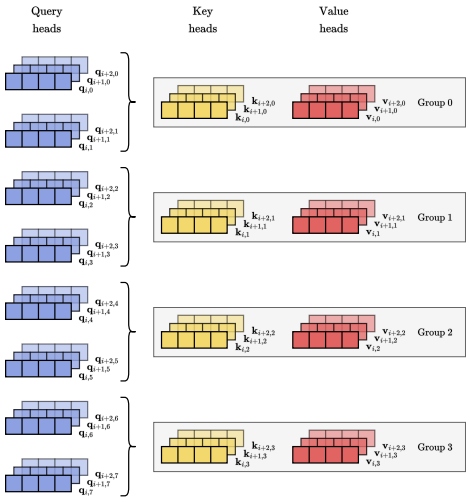


Figure 2.13: Illustration of how query, key, and value heads are associated in grouped-query attention. For each position of the input sequence $\{i, i+1, \dots\}$, queries are grouped into G sets, where each group shares a common set of key and value vectors. In particular, considering the j-th head and the k-th head belonging to the same group g, they operate respectively on queries $\{\mathbf{q}_{i,j}, \mathbf{q}_{i+1,j}, \dots\}$ and $\{\mathbf{q}_{i,k}, \mathbf{q}_{i+1,k}, \dots\}$, while sharing the same keys $\{\mathbf{k}_{i,g}, \mathbf{k}_{i+1,g}, \dots\}$ and values $\{\mathbf{v}_{i,g}, \mathbf{v}_{i+1,g}, \dots\}$.

Encoder-Decoder structure

As previously introduced, the Transformer architecture, presented in *Attention is All You Need*, relies on a modular encoder-decoder structure. Both the encoder and decoder consist of multiple identical blocks arranged sequentially. This hierarchical design allows the model to progressively refine its representation of the input sequence, capturing increas-

ingly complex dependencies among tokens with each successive layer.

An important component of this architecture lies in the embedding layer, which is shared by both the encoder and decoder. This layer maps discrete input tokens into continuous token embeddings. These embeddings are essential to provide the model with a rich, context-aware input representation. The weights of the embedding layer are trainable and updated during the training process of the model. For the encoder, the embeddings of the input sequence are computed and passed to the first encoder block. Similarly, for the decoder, embeddings of the target sequence (or partial sequences during inference) are computed before being fed into the decoder stack.

The Encoder

As illustrated in Figure 2.8, each encoder block comprises two primary sub-layers:

1. **Multi-Head Self-Attention** - The input embeddings are processed through a self-attention mechanism (Section 2.4.2), which evaluates the importance of each token in the sequence relative to all others and produces a new representation for each position. For the first Transformer block, the inputs to this layer are the embeddings of the words in the sequence. In subsequent blocks, the inputs consist of the outputs generated by the preceding layers.

2. **Position-Wise Feed Forward Neural Network** - Following the self-attention mechanism, each token is independently processed through a feed forward network.

   The original Transformer model employed a two-layer multilayer perceptron, comprising an up-projection layer that maps the embedding to a higher-dimensional space, followed by a down-projection layer that restores it to the original hidden dimension. Modern architectures often enhance this design by incorporating a three-layer configuration. This setup includes an up-projection layer and a gating layer whose outputs are combined, followed by a final down-projection layer.

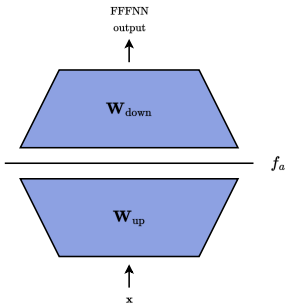
   For a simple two-layer FFNN, the transformation applied to an input token $\mathbf{x} \in \mathbb{R}^{d_h}$, where d_h is the embedding dimension, is expressed as:

$$\mathbf{FFNN}(\mathbf{x}) = f_a\left(\mathbf{x} \cdot \mathbf{W}_{\text{up}}^T\right) \cdot \mathbf{W}_{\text{down}}^T$$

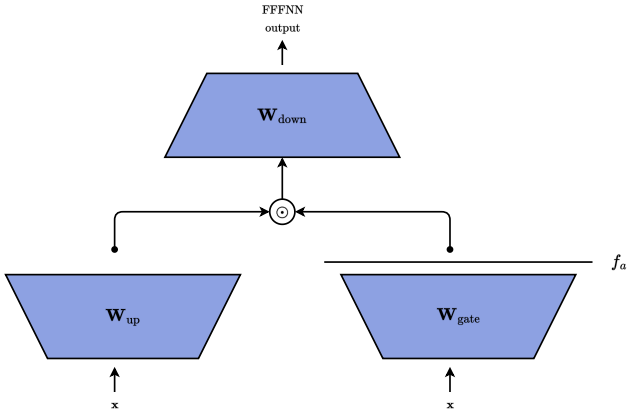
   In the case of a three-layer FFNN, the transformation is defined as:

$$\mathbf{FFNN}(\mathbf{x}) = \left(f_a\left(\mathbf{x} \cdot \mathbf{W}_{\text{gate}}^T\right) \circ \left(\mathbf{x} \cdot \mathbf{W}_{\text{up}}^T\right)\right) \cdot \mathbf{W}_{\text{down}}^T$$

Here, f_a denotes a non-linear activation function, and $\circ$ represents the element-wise Hadamard product. Although bias terms can be added after each linear transformation to enhance flexibility, they are typically omitted in modern Transformer architectures.



(a) Diagram of a feed forward neural network module with two layers inside Transformer blocks.



(b) Diagram of a feed forward neural network module with three layers inside Transformer blocks.

Figure 2.14: Diagrams of feed forward neural networks within Transformer blocks.

- **Residual Connections** - Residual connections wrap around each sub-layer, allowing layers to learn residual modifications to the identity mapping rather than performing complete transformations of the input. This simplifies the learning process and facilitates the flow of gradients during backpropagation, mitigating the vanishing gradient problem. By preserving information across layers, residual connections enable the effective training of Transformers and enhance optimization stability.
- **Layer Normalization** - In the original Transformer architecture, layer normalization is applied after each layer, specifically following the summation of the output of the attention mechanism and the output of the feed forward network with the residual connection. This technique is pivotal for stabilizing and accelerating the

training process by normalizing inputs across features for each individual training example in a batch. Unlike batch normalization, which operates across the batch dimension, layer normalization ensures that each element in a sequence is normalized independently. In contemporary models, layer normalization is frequently applied before each layer, improving gradient flow and ensuring stability during the training of extremely large models.

Given an input vector $\mathbf{x} = [x_1, x_2, \ldots, x_{d_h}]$ with d_h dimensions, representing a token in an input sequence for a Transformer model, layer normalization operates as follows:

1. Compute the mean and variance: The mean μ and variance σ^2 of the input vector are computed using the equations:

$$\mu = \frac{1}{d_h} \sum_{i=1}^{d_h} x_i \quad \text{(mean)}$$

$$\sigma^2 = \frac{1}{d_h} \sum_{i=1}^{d_h} (x_i - \mu)^2 \quad \text{(variance)}$$

2. Normalize the input: Each element x_i is normalized by subtracting the mean and dividing by the standard deviation:

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

   where ϵ is a small constant added to avoid division by zero.

3. Scale and shift: The normalized input is then scaled and shifted using learnable parameters γ_i (scaling factor) and β_i (bias term):

$$y_i = \gamma_i \hat{x}_i + \beta_i$$

   Here, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are trainable vectors that have the same size as $\mathbf{x}$.

The entire operation can be summarized in the equation:

$$\mathbf{y} = \boldsymbol{\gamma} \cdot \frac{\mathbf{x}^T - \mu}{\sqrt{\sigma^2 + \epsilon}} + \boldsymbol{\beta}$$

The Decoder

The decoder shares a similar architecture with the encoder but incorporates a modified sub-layers tailored to its generative purpose:

1. **Masked Multi-Head Self-Attention.** This layer adapts the self-attention mechanism found in encoder blocks by incorporating a causal mask that prevents the decoder from attending to future tokens. This adjustment is essential during training, as it simulates the model's inference behavior, where tokens are generated sequentially in an autoregressive fashion. At each step of generation, the model relies solely on tokens from preceding steps, ignoring future ones.

   The causal mask $\mathbf{M} \in \mathbb{R}^{l \times l}$ is a triangular matrix defined as:

   $$M_{ij} = \begin{cases} 0 & \text{if } j \leq i \\ -\infty & \text{if } j > i \end{cases}$$

   By incorporating this mask, the attention scores are adjusted as:

   $$\mathbf{A} = \text{softmax}\left(\frac{\mathbf{Q} \cdot \mathbf{K}^\top}{\sqrt{d_k}} + \mathbf{M}\right)$$

   Adding $-\infty$ ensures that the softmax function assigns zero probability to masked positions. The resulting masked self-attention output is then computed as:

   $$\textbf{Masked Self-Attention}\,(\mathbf{X} \mid \mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathbf{A} \cdot \mathbf{V}$$

2. **Encoder-Decoder Attention.** Following the masked self-attention, the decoder performs cross-attention to integrate information coming from the encoder. This mechanism allows the decoder to selectively focus on relevant parts of the encoded input sequence. The cross-attention process mirrors the steps of self-attention but uses the encoder states to compute keys and values, while the decoder states provide the queries.

The decoder incorporates FFNNs, residual connections, and layer normalization modules, which operate in the same way as in the encoder, providing equivalent functionality and contributing to the overall stability and effectiveness of the model.

Numerous variants of the original Transformer model have since been developed, all building upon the foundational innovations introduced in *Attention is All You Need*.

Encoder-only Models

An encoder-only model is a type of Transformer architecture composed solely of a stack of encoder layers, completely omitting the decoder component. BERT is a prominent example of this design. These models utilize a traditional bidirectional self-attention mechanism, enabling them to attend to all tokens in the input sequence simultaneously, considering both preceding and following tokens. This feature allows encoder-only models to excel in tasks that require a deep understanding of input text. By generating rich contextual embeddings, these models capture the meaning of each token in relation to the entire input sequence, providing a nuanced understanding of its surrounding context.

During pre-training, encoder-only models typically employ masked language modeling as their learning objective. In this approach, sentences with specific tokens masked are presented to the model, which then predicts the masked tokens based on the unmasked context.

Unlike architectures designed for text generation, encoder-only models are primarily used for comprehension-focused tasks such as text classification, token classification, and extractive question answering.

Decoder-Only Models

Decoder-only models represent a specialized variant of Transformer architectures, consisting exclusively of a stack of Transformer decoder layers and omitting any encoder component. These models are particularly well-suited for generative tasks, such as language modeling and text completion, due to their sequential token processing and ability to predict the next token in a sequence. Furthermore, their ability to capture contextual relationships within sequences allows them to perform effectively in certain understanding tasks.

A hallmark of decoder-only models is their exclusive reliance on masked self-attention, restricting each token's access to only the preceding tokens in the sequence. This causal structure, as implemented in the original Transformer model introduced in *Attention is All You Need*, is essential for autoregressive tasks, enabling the model to predict the next token based on prior context. Through the sequential addition of tokens, decoder-only models can produce coherent and contextually relevant text.

Training these models typically involves a next-token prediction objective, where the model learns to maximize the likelihood of each token in a sequence given its preceding tokens. This approach allows decoder-only models to effectively capture both local and

global dependencies, making them versatile for a wide range of natural language processing tasks.

The simplicity and effectiveness of their design have established decoder-only models as the dominant choice in modern NLP, powering many state-of-the-art systems for both text generation and comprehension.

Positional Encoding

While self-attention excels at modeling relationships among tokens, it is inherently permutation-invariant; the attention mechanism produces identical results regardless of the order in which tokens appear. However, in natural language, word order is crucial, as rearranging words in a sentence can drastically change its meaning. To address this limitation, positional encoding is introduced. This mechanism encodes the position of each token within the sequence, providing the model with the ability to distinguish between different orders and capture the sequential structure of the data.

Positional encodings are integrated with token embeddings through element-wise addition. Specifically, each token's embedding vector, which encodes its semantic properties, is enriched with a corresponding positional encoding vector that conveys its position in the sequence. This combination imbues the model with both semantic and positional context, ensuring it can process inputs in a manner sensitive to word order.

Transformers employ various strategies for positional encoding, each tailored to specific use cases and offering unique advantages. These methods are explored in the subsequent discussion.

Sinusoidal Positional Encoding

Sinusoidal positional encoding was the original positional encoding proposed in the Transformer paper. Sinusoidal encodings use fixed sine and cosine functions with different frequencies to assign unique, continuous positional information to each token in the sequence.

Considering d_h as the dimensionality of the embedding. For each position pos of the tokens in the sequence and for i ranging from 0 to $\frac{d_h}{2} - 1$, the positional encoding is defined as:

Considering d_h as the dimensionality of the embeddings, the positional encoding for a token at position pos in the sequence is defined as follows for i ranging from 0 to $\frac{d_h}{2} - 1$:

$$SPE(pos, 2i) = \sin\left(\frac{\text{pos}}{10000^{2i/d_h}}\right),$$
$$SPE(pos, 2i+1) = \cos\left(\frac{\text{pos}}{10000^{2i/d_h}}\right).$$

In this formulation, even indices of the embedding dimension ($2i$) are computed using the sine function, while odd indices ($2i + 1$) are computed using the cosine function.

Sinusoidal encoding does not require additional learned parameters, which can save on memory and computation. This method supports arbitrary sequence lengths, the model can generalize to longer or shorter sequences since these encoding follow a continuous pattern. The model can infer relative distances between tokens based on the frequency patterns in the sine and cosine waves, which makes it useful for capturing hierarchical relationships in sequences.

Learned Positional Encoding

In learned positional encoding, the model assigns a unique embedding vector to each position in the sequence, which is optimized during training in the same manner as input token embeddings. This approach is also known as absolute positional encoding.

Each position is represented by a distinct embedding vector that is randomly initialized and refined throughout training alongside the other parameters of the model.

Learned positional encodings are tailored to the specific task the model is trained on, often resulting in improved performance compared to fixed encodings for certain applications. However, this approach comes with some limitations. The embeddings are restricted to a predefined maximum sequence length, limiting the model's ability to generalize to sequences longer than those encountered during training. Additionally, learned positional encodings introduce extra parameters and increase memory requirements, particularly when dealing with long sequences.

Relative Positional Encoding

Relative positional encoding offers an alternative approach to representing token order by focusing on the distance between tokens rather than their absolute positions within a sequence. This method can be implemented in various ways, with one common technique involving the introduction of distance-dependent bias terms during self-attention computations. By encoding relative distances, each token gains contextual awareness of its surroundings based on relational rather than positional information.

This paradigm enables models to generalize more effectively to sequences longer than those encountered during training, as it emphasizes relative relationships over fixed positions. Such flexibility proves particularly advantageous in tasks where the relative order of tokens holds greater importance than their absolute placement. Additionally, relative positional encoding can enhance computational efficiency when handling longer sequences, especially in Transformer architectures designed for long-range attention.

Rotary Positional Encoding

Rotary Positional Encoding (RoPE) [51] is an approach to positional encoding that employs rotational transformations on embedding vectors rather than incorporating positional embeddings through addition or learned parameters. This technique has been prominently utilized in models such as GPT-NeoX and Llama.

RoPE operates by rotating the embedding vectors through angles that progressively increase with each token's position in the sequence. These position-dependent rotations elegantly encode relative positional information, enabling the model to capture relationships between tokens without the need for augmenting the dimensionality of embeddings. This design enhances the model's ability to understand positional context while maintaining computational efficiency.

Given an input embedding vector $\mathbf{x} \in \mathbb{R}^{d_h}$, where d_h is even, the RoPE transformation is expressed as:

$$\mathbf{RoPE}(\mathbf{x}) = \mathbf{x} \cdot \mathbf{M}(pos_{\mathbf{x}} \cdot \boldsymbol{\theta})^T,$$

where the components are defined as follows:

- $\boldsymbol{\theta} \in \mathbb{R}^{d_h/2}$ is a vector of angular frequencies, typically defined as $\theta_i = 10000^{-2i/d_h}$, with i representing the index of a pair of dimensions, ranging from 0 to $d_h/2 - 1$.
- $pos_{\mathbf{x}}$ represents the position of the token $\mathbf{x}$ within the input sequence.
- $\mathbf{M}(pos_{\mathbf{x}} \cdot \boldsymbol{\theta})$ is a rotation matrix of size $\mathbb{R}^{d_h \times d_h}$, defined block-wise. Each block rotates a two-dimensional subspace by its corresponding angle $pos_{\mathbf{x}} \cdot \theta_i$.

The structure of the rotation matrix $\mathbf{M}$ is defined as:

$$\mathbf{M}(pos_{\mathbf{x}} \cdot \boldsymbol{\theta}) = \begin{bmatrix} \cos(pos_{\mathbf{x}} \cdot \theta_0) & -\sin(pos_{\mathbf{x}} \cdot \theta_0) & 0 & 0 & \cdots \\ \sin(pos_{\mathbf{x}} \cdot \theta_0) & \cos(pos_{\mathbf{x}} \cdot \theta_0) & 0 & 0 & \cdots \\ 0 & 0 & \cos(pos_{\mathbf{x}} \cdot \theta_1) & -\sin(pos_{\mathbf{x}} \cdot \theta_1) & \cdots \\ 0 & 0 & \sin(pos_{\mathbf{x}} \cdot \theta_1) & \cos(pos_{\mathbf{x}} \cdot \theta_1) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

.

RoPE is inherently scalable to longer sequences, as it does not depend on a predefined maximum sequence length. This flexibility allows the model to effectively capture relative positional information, making it particularly advantageous for tasks that rely on understanding token-to-token relationships independent of their absolute positions.

When paired with self-attention, positional encoded enhances the Transformer's ability to model both token relationships and their sequential context. This synergy contributes to state-of-the-art performance across a wide array of natural language processing tasks.

2.4.3. Large Language Models

The Transformer architecture represents a monumental leap in natural language processing, providing the foundation for the development of LLMs [30] [31]. Large language models are a class of artificial intelligence models specifically designed to process and generate human language. By leveraging the self-attention mechanisms and parallelization inherent in Transformers, LLMs excel at capturing complex patterns and dependencies within text, enabling them to understand and generate human language with remarkable accuracy. Their success also hinges on the availability of vast and diverse training corpora, spanning a wide range of topics, languages, and styles, which allows them to develop a generalized understanding of language. This adaptability makes LLMs highly effective for a variety of tasks, including language translation, summarization, question answering, content generation, and more.

Models such as BERT, GPT, Llama, Gemma, Mistral, and others exemplify the power of language models, demonstrating their ability to comprehend context, infer meaning, and produce coherent, contextually relevant responses.

Mathematical Definition of Language modeling

Language modeling is a fundamental task in natural language processing, focused on predicting the likelihood of a sequence of tokens, words or sub-words, in a given language. Formally, let $\mathbf{x} = [x_0, x_1, \ldots, x_T]$ denote a sequence of word tokens, where T represents the sequence length and x_t is the token at position t. The objective of a language model is to estimate the probability distribution $\mathrm{P}(\mathbf{x})$ over all possible token sequences within the language.

By leveraging the chain rule of probability, this joint probability distribution can be factorized as:

$$P(\mathbf{x}) = P(x_0, x_1, \ldots, x_T) = \prod_{t=0}^{T} P(x_t \mid x_0, x_1, \ldots, x_{t-1})$$

where $P(x_t \mid x_0, x_1, \ldots, x_{t-1})$ is the conditional probability of token x_t given the preceding tokens $x_0, x_1, \ldots, x_{t-1}$.

Language models are trained to maximize the likelihood of sequences in a given training dataset. The training objective, for a dataset $\mathcal{D}$ containing sequences $\{\mathbf{x}^i\}_{i=0}^{N}$, is to minimize the negative log-likelihood:

$$\mathcal{L} = -\frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T_i} \log P(x_t^i \mid x_0^i, x_1^i, \ldots, x_{t-1}^i)$$

where T_i is the length of the i-th sequence in the dataset.

Pretrained Models

In general, large language models are first pretrained on massive and diverse corpora to learn general-purpose language representations. This pretraining phase involves self-supervised learning, where the model predicts missing or subsequent tokens in a sequence without requiring explicit labeled data. Techniques such as masked language modeling (e.g. in BERT) or autoregressive modeling (e.g. in GPT) are commonly employed to optimize the model for understanding language patterns, structures, and contextual relationships.

Pretrained models can then be adapted for specific tasks through various approaches, each tailored to different levels of customization and computational efficiency:

- **Prompting**: This method involves crafting input queries to guide the model's

behavior without modifying its internal parameters. Prompting is particularly useful when the aim is to exploit pretrained model's capabilities directly, avoiding the need for expensive additional training. It allows users to utilize the model out of the box, making it an efficient and practical solution for tasks with minimal labeled data or limited computational resources.

- **Transfer Learning**: Transfer learning extends the pretrained model's capabilities to new domains or tasks by introducing domain-specific corpora. This intermediate step involves training additional task-specific components while keeping the model's core parameters unchanged. This method is particularly effective when the target domain differs slightly from the model's original training context.
- **Fine-Tuning**: Fine-tuning refines the model by updating its parameters using labeled datasets tailored to a specific application. Unlike transfer learning, fine-tuning adjusts the model's internal weights, allowing it to learn the nuances of a particular task. This approach is ideal for achieving high accuracy in specialized domains or tasks with sufficient labeled data.

Prompting

Prompting is a lightweight and versatile method of task adaptation that involves crafting input queries to direct a model's behavior without modifying its internal parameters. It maximizes efficiency by leveraging the pretrained capabilities of the model and is especially suited for applications where computational or data resources are limited. Depending on the complexity and requirements of the task, prompting can take several forms:

1. **Direct Prompting (Zero-Shot Learning)**: This approach uses precise, explicit instructions to define the task the model has to solve. Without any task-specific examples, the model relies solely on its general language understanding.
    - *Example*: Translate the word dog into Italian.
2. **Contextual Prompting**: Contextual prompting enhances task performance by providing examples within the input, illustrating patterns or expectations. This method can be subdivided into:
    (a) **One-Shot Learning**: The model is given a single example to clarify the task's nature and expected output.
        - *Example*: Translate in words into Italian: Cat → Gatto; Dog → *?.
    (b) **Few-Shot Learning**: A small number of examples are included to demon-

strate the task's specific requirements and patterns, further improving the model's performance.

- *Example*: Translate in words into Italian: Cat → Gatto; Hamster → Criceto; Rabbit → Coniglio; Dog → *?.

These paradigms capitalize on the generalization abilities of LLMs, enabling efficient adaptation to new tasks with zero computational overhead. By intelligently structuring input queries, users can unlock the potential of the model for a diverse range of applications, even in resource-constrained environments.

Transfer Learning

Transfer learning is the process of applying a pretrained model to solve a related, yet distinct, task. In the context of deep learning, this approach typically involves harnessing the general feature extraction capabilities of the pretrained model and integrating a task-specific output layer, often referred to as a "head", on top of its architecture. This specialized head is trained exclusively on the dataset and requirements of the target task, while the remaining layers of the pretrained model remain fixed.

The newly added component is designed to handle the unique demands of the specific task, leveraging the robust representation capabilities of the pretrained model. Transfer learning demonstrates significant efficacy in scenarios with limited task-specific data or when the knowledge encoded in the pretrained model aligns closely with the problem being addressed.

Fine-Tuning

Fine-tuning involves refining a pretrained model by training it further using a smaller dataset designed for the target task. This approach allows the model to specialize in a particular application by updating its internal parameters, adapting to the unique characteristics of the task while preserving the general knowledge acquired during pretraining.

While full fine-tuning, where all layers of the pre-trained model are modified, enables comprehensive adaptation to the new task, this procedure can be computationally demanding due to size of large language models. To mitigate these challenges, several strategies have been developed:

1. **Partial Fine-Tuning**: A significant portion of the parameters of the model is kept frozen, with only a subset, such as the final layers, being updated. This method reduces computational overhead while retaining the general knowledge embedded

in the pretrained model.

2. **Adapter-Based Fine-Tuning**: Lightweight, task-specific modules known as adapters are introduced into the architecture. These adapters are trained independently while the core parameters remain unchanged, enabling efficient and flexible adaptation. This method is computationally efficient and modular, as adapters can be added or swapped out for different tasks without altering the pretrained LLM. Examples include techniques like LoRA [27], Adaptive Low-Rank Adaptation (AdaLoRA) [58], and Vector-based Random Matrix Adaptation (VeRA) [32].

Fine-tuning is particularly effective when the target task diverges significantly from the original objectives of the pretrained model and when sufficient task-specific data is available to support the adaptation process. By aligning the capabilities of deep architectures with the requirements of new domains, fine-tuning ensures optimized task-specific performance while balancing computational efficiency and specialization.

Adapter-Based Fine-Tuning: Efficient Fine-Tuning

Adapter-based fine-tuning offers a resource-efficient approach to adapt large pre-trained models to specific tasks. This technique integrates lightweight modules, known as adapters, within the architecture. Unlike traditional fine-tuning, which updates all model parameters, adapter-based methods optimize only the introduced weights, significantly reducing computational and storage requirements.

Low-Rank Adaptation of Large Language Models

A prominent technique within adapter-based fine-tuning is Low-Rank Adaptation (LoRA) [27], which effectively balances performance and efficiency by representing fine-tuning parameter updates as low-rank matrix modifications.

LoRA operates by introducing trainable low-rank matrices, the adapters, while keeping the original weights frozen. These matrices are optimized during fine-tuning, enabling task-specific adaptation without directly altering the base parameters.

For a dense weight matrix $\mathbf{W}_0$ of dimensions $m \times n$ in the model, LoRA approximates the delta matrix learnt during fine-tuning as a product of two smaller matrices:

$$\mathbf{\Delta W} \approx \mathbf{A} \cdot \mathbf{B}$$

where $\mathbf{A} \in \mathbb{R}^{m \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times n}$, with typically $r \ll \min(m, n)$.

By constraining r to be much smaller than $\min(m, n)$, LoRA drastically reduces the number of trainable parameters. The model weights $\mathbf{W}$ are then represented as:

$$\mathbf{W} = \mathbf{W}_0 + \mathbf{\Delta W} = \mathbf{W}_0 + \mathbf{A} \cdot \mathbf{B}$$

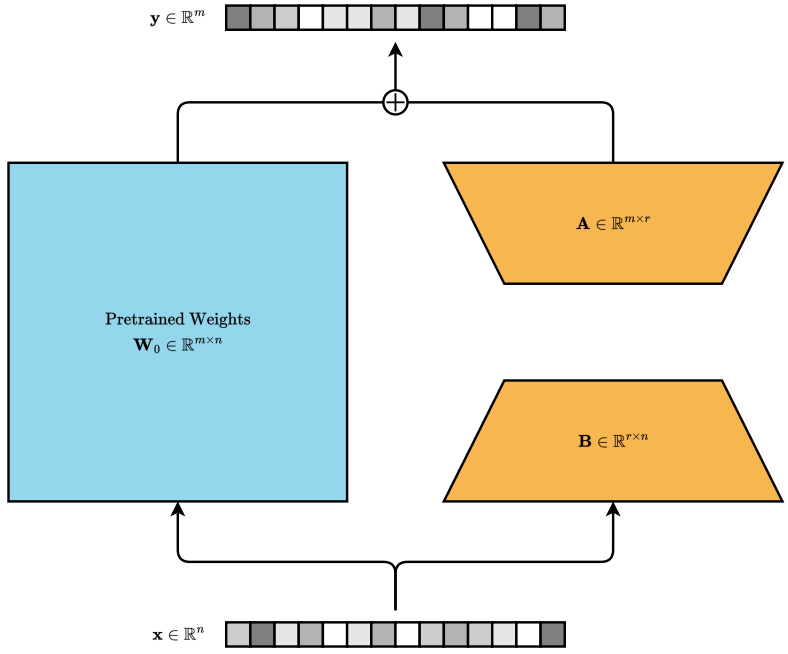


Figure 2.15: Diagram of the LoRA method.

LoRA fine-tuning drastically reduces the number of trainable parameters, making the process both faster and more memory-efficient compared to full fine-tuning. This efficiency is particularly beneficial in resource-constrained scenarios. By keeping the base model unchanged, task-specific adapters can be stored and shared independently, minimizing storage requirements. The modular nature of LoRA adapters allows for seamless switching between tasks without the need to retrain the base model, offering significant flexibility. Moreover, despite its resource efficiency, LoRA consistently delivers performance that is competitive with traditional full fine-tuning across a wide range of tasks.

Instruction Tuning

Instruction tuning is a fine-tuning technique that adapts large language models to better interpret and execute complex, multi-step instructions presented in natural language. By training on datasets containing diverse task instructions and their outputs, models become proficient at understanding a wide variety of directives and generating accurate,

task-specific responses. These datasets typically include tasks such as translation, text summarization, or answering detailed queries, all framed as natural language instructions.

This approach builds on the general linguistic knowledge of pretrained models, while tailoring them for specific task scenarios. Instruction tuning promotes structured reasoning, enabling models to generalize effectively across a wide range of applications. By aligning models more closely with human intent, this technique enhances usability in real-world settings.

A notable distinction within instruction tuning lies in the specialization for different use cases. Two prominent examples are the fine-tuning of instruction-following agents and the development of conversational agents, or chatbots.

Instruction-Following Agents. Fine-tuning using instructions is designed to develop instruction-following agents, which excel at executing precise tasks guided by natural language directives. These agents are optimized to handle diverse instructions, providing accurate outputs across a wide spectrum of domains. This type of specialization emphasizes task completion and structured reasoning, often prioritizing adherence to specific user inputs.

Chatbot Fine-Tuning. Chatbot tuning, on the other hand, focuses on optimizing LLMs for conversational tasks, transforming them into interactive systems capable of simulating human-like dialogue. Fine-tuning on conversational datasets enables these models to understand user intents, maintain contextual coherence, and generate natural, context-aware responses. Chatbots are widely used in domains such as customer service, healthcare, and education, where engaging and contextually relevant communication is paramount.

To further enhance chatbot performance, techniques like Reinforcement Learning from Human Feedback (RLHF) are often employed. RLHF enables iterative improvement based on human evaluators' feedback, ensuring that chatbots align with human expectations and values and deliver socially and ethically appropriate responses. This feedback loop contributes to seamless conversational experiences, making chatbots more effective across diverse applications.

2.5. Large Language Models Compression

Since their initial development following the introduction of the Transformer architecture, large language models have grown exponentially in size, consistently achieving per-

formance improvements with increasing parameter counts. However, their considerable computational and memory requirements pose significant challenges for deployment in resource-constrained environments. This has driven research into model compression techniques, which aim to create smaller, more efficient versions of large language models while preserving their expressive power. These techniques seek to optimize memory, computational and storage efficiency without compromising the ability of models to understand and generate language.

LLMs encapsulate large collections of numerical parameters that encode both linguistic capabilities and knowledge acquired from training on vast datasets. While the exact relationship between these parameters and the model's ability to generate coherent, meaningful language remains only partially understood, compression techniques operate on the premise that certain parameters contribute negligibly to performance and may represent redundant or superfluous information. By identifying and eliminating these inefficiencies, computational overhead can be reduced while preserving model quality.

In the context of this thesis, which focuses on identifying redundancy within Transformer modules and leveraging it for compression, a thorough understanding of existing compression techniques is essential. These methods, though diverse in their underlying principles, share a common objective: to compact the knowledge encoded in pretrained models into a smaller, more efficient representation.

Some compression techniques necessitate training or fine-tuning to adjust model parameters during or after compression. This iterative refinement process enables the model to adapt to the constraints introduced by compression, often enhancing fidelity to the original performance while, at the same time, incurring substantial computational costs and prolonged processing time.

Several strategies have been developed to compress deep neural networks, with the most prominent being quantization, knowledge distillation, pruning, and low-rank approximation. The following sections provide a detailed examination of these techniques, analyzing their underlying principles, advantages, and limitations in the context of deep neural architectures.

2.5.1. Quantization

Quantization [22] [41] [60] is a technique that reduces the numerical precision of a model's parameters and, in some cases, activations to lower-bit representations such as 16-bit floating-point, 8-bit integers, or even binary values. This process decreases memory usage, reduces model size, and improves computational efficiency, particularly on hardware

optimized for low-precision arithmetic. Quantization can be applied using Post-Training Quantization (PTQ) [17] [35], where a pre-trained model is converted to a lower precision without additional training, or through Quantization-Aware Training (QAT) [38] [11], where the model is transformed to a lower-bit representation and retrained to compensate for the performance loss introduced by quantization.

Quantization offers significant efficiency gains with minimal degradation in model performance when applied carefully. It also enables models to take advantage of modern hardware, such as GPUs and Tensor Processing Units (TPUs), that are optimized for low-precision operations. However, this method can face challenges, including numerical instability, particularly in layers that are highly sensitive to weight perturbations. Additionally, quantization may not perform as effectively for tasks requiring fine-grained numerical precision, such as those involving highly sensitive computations. Techniques like post-training quantization and quantization-aware training help mitigate these issues, ensuring robust model performance even after compression.

2.5.2. Knowledge Distillation

Knowledge distillation [24] [60] is a technique where a smaller student model is trained to emulate the behavior of a larger teacher model. In this process, the student learns not only from the ground truth labels but also from the predictions generated by the teacher. These predictions typically include soft probabilities for various output classes, offering deeper insights into the reasoning process employed by the teacher. By leveraging this additional information, the student model can achieve a significant portion of the performance and generalization capabilities of the teacher, despite being significantly smaller in size. Various adaptations and extensions of this paradigm exist, each tailored to optimize different aspects of model compression and efficiency.

This method is particularly effective in creating compact models that retain the knowledge and performance of their larger counterparts. Knowledge distillation also provides flexibility, as the student model can be of a completely different architecture, making it suitable for deployment in resource-constrained environments. However, the training process for the student model can be computationally expensive, as it requires generating predictions from the teacher model for the entire dataset. Additionally, the success of knowledge distillation depends on careful optimization to ensure that the student model can adequately replicate the behavior of the teacher, particularly for task-specific requirements.

2.5.3. Pruning

Pruning is a compression technique aimed at reducing the size and computational complexity of a model by removing unnecessary or less significant components, such as weights, neurons, or even entire layers. The goal is to maintain the core functionality and structure of the model while discarding redundant or non-essential parameters.

This technique can be implemented in several forms. Unstructured pruning removes individual weights based on predefined importance criteria, often resulting in sparse weight matrices. However, these sparse structures may require specialized hardware or software to achieve efficient execution. On the other hand, structured pruning eliminates larger, self-contained components of the architecture, such as matrix sections, attention heads, or even entire layers or blocks. This approach produces compact models that are more easily deployable on standard hardware. A hybrid approach, referred to as semi-structured pruning, removes parameters based on specific patterns. This method strikes a balance between the flexibility of unstructured pruning and the regularity of structured pruning, while also leveraging those patterns to ensure compatibility with hardware optimized for such configurations.

Pruning is particularly effective in over-parameterized models, where a significant proportion of parameters contribute minimally to overall performance. It reduces both model size and inference time, making the model more efficient without compromising its essential capabilities. Various pruning criteria are proposed in the literature, tailored to specific scenarios or architectures. Careful selection of these criteria and the corresponding pruning thresholds is vital to ensure that the compressed model achieves meaningful resource savings while maintaining its accuracy and functionality.

Structured Pruning

Structured pruning involves removing predefined components of a model, such as rows, columns, channels, attention heads, or even entire transformer blocks. This approach is inherently compatible with hardware accelerators, as it preserves the dense matrix operations required for efficient computation on GPUs and TPUs. By focusing on structured units, this method avoids the irregular sparsity patterns that unstructured pruning introduces.

Unstructured Pruning

Unstructured pruning operates at a fine-grained level, removing individual weights based on their importance. A straightforward approach involves thresholding parameters by magnitude, where lower-magnitude weights are deemed less critical and subsequently pruned [59] [19]. While this method offers flexibility and scalability, its application to large language models often leads to significant quality degradation.

A more sophisticated and effective technique is Optimal Brain Surgeon (OBS) [34] [23], which removes weights based on their impact on the loss function. By addressing some of its computational limitations, subsequent methods such as WoodFisher [48] and Optimal Brain Compression (OBC) [15] have refined this approach, further demonstrating the effectiveness of pruning strategies guided by loss sensitivity. Expanding on these advancements, techniques like SparseGPT [16] and Wanda [52], discussed in Section 3.1.2, enhance these ideas by offering more efficient and practical compression methods.

However, the irregular sparsity patterns introduced by unstructured pruning are less compatible with standard hardware accelerators, often requiring modifications or specialized libraries or hardware to realize meaningful computational speedups.

Low-rank Factorization

Low-rank factorization is a compression technique that reduces the memory and computational cost of deep models by replacing large weight matrices with products of smaller, lower-rank matrices. In LLMs, this reduces the number of parameters while preserving the model's expressive power and performance.

By decomposing weight matrices, low-rank factorization retains their most informative components while discarding redundancy. Various decomposition techniques can be used to obtain low-rank approximations, each offering different computational trade-offs and varying effectiveness in preserving model performance.

Unlike pruning, which removes parameters, it restructures weight matrices into a more compact yet dense form. This maintains compatibility with standard hardware accelerators, ensuring efficient matrix multiplications without altering the computational structure.

2.5.4. Matrix Factorization Techniques

Matrix factorization, or matrix decomposition, is a foundational concept in linear algebra that refers to the process of expressing a matrix as the product of two or more matrices.

Different factorization techniques vary in their mathematical principles, computational efficiency, and suitability for specific model architectures and applications. This section provides an overview of key decompositions in mathematical literature and their relevance to model compression.

Matrix decompositions are particularly valuable because they allow for the isolation of important characteristics of the matrix, such as rank, singular values, or eigenvalues. These properties provide interesting insights into the underlying behavior of the matrix. As such, matrix factorization is not merely an abstract theoretical tool but a practical technique used to streamline operations and obtain clearer interpretations of data or models represented by matrices.

The choice of decomposition depends on the structure of the target matrix and the intended use of the factorization. Certain decompositions show specific properties, such as symmetry, orthogonality, or triangularity, that make the, useful in particular contexts. Key decompositions such as LU decomposition, QR decomposition, eigendecomposition, singular value decomposition and Hadamard decomposition are central to modern mathematical analysis, each with its own unique set of properties, applications, and theoretical foundations.

LU Decomposition

LU decomposition represents a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the product of two triangular matrices, $\mathbf{L}$ and $\mathbf{U}$, such that:

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{U}.$$

Here, $\mathbf{L} \in \mathbb{R}^{n \times n}$ denotes a lower triangular matrix and $\mathbf{U} \in \mathbb{R}^{n \times n}$ is an upper triangular matrix. The decomposition can be generalized to rectangular matrices.

QR Decomposition

QR decomposition expresses a matrix $A \in \mathbb{R}^{m \times n}$ as the product of a matrix $\mathbf{Q}$ and a matrix $\mathbf{R}$, such that:

$$\mathbf{A} = \mathbf{Q} \cdot \mathbf{R},$$

where $\mathbf{Q} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix, namely it satisfies $\mathbf{Q}^T \cdot \mathbf{Q} = \mathbf{I}$, where $\mathbf{I} \in \mathbb{R}^{m \times m}$ denotes the identity matrix, and $\mathbf{R} \in \mathbb{R}^{m \times n}$ is an upper triangular matrix.

Cholesky Decomposition

Cholesky decomposition is a specialized decomposition applicable to positive definite matrices $\mathbf{A} \in \mathbb{R}^{n \times n}$. It represents $\mathbf{A}$ as the product of a lower triangular matrix $\mathbf{L}$ and its transpose, such that:

$$\mathbf{A} = \mathbf{L} \cdot \mathbf{L}^T,$$

where $\mathbf{L} \in \mathbb{R}^{n \times n}$ is a lower triangular matrix with strictly positive diagonal entries.

Properties:

- **Positive Definiteness**: Cholesky decomposition is defined if and only if $\mathbf{A}$ is symmetric and positive definite.
- **Uniqueness**: The decomposition is unique when $\mathbf{A}$ is positive definite.
- **Computational Efficiency**: Compared to other matrix factorizations such as LU or QR, Cholesky decomposition is computationally more efficient due to its reliance on symmetry and positive definiteness.

Eigendecomposition

Eigendecomposition, also referred to as spectral decomposition, expresses a square matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ as the product of a matrix of eigenvectors, $\mathbf{V}$, and a diagonal matrix of eigenvalues, $\mathbf{\Lambda}$, as follows:

$$\mathbf{A} = \mathbf{V} \cdot \mathbf{\Lambda} \cdot \mathbf{V}^{-1}.$$

In this formulation, $\mathbf{V} \in \mathbb{R}^{n \times n}$ is a matrix whose columns correspond to the eigenvectors of $\mathbf{A}$, while $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the eigenvalues of $\mathbf{A}$ along its diagonal.

Properties:

- **Diagonalizability**: A matrix is diagonalizable if, and only if, it possesses a sufficient number of linearly independent eigenvectors to construct the matrix $\mathbf{V}$. This condition is equivalent to the matrix admitting an eigendecomposition.
- **Symmetric Matrices**: For symmetric matrices, the eigenvectors can always be chosen to be orthogonal, making $\mathbf{V}$ an orthogonal matrix. In this case, the matrix satisfies $\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T$.

Singular Value Decomposition

Singular Value Decomposition (SVD) expresses a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ as the product of three matrices, $\mathbf{U}$, $\mathbf{\Sigma}$ and $\mathbf{V}$, such that:

$$\mathbf{A} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T.$$

In this decomposition:

- $\mathbf{U} \in \mathbb{R}^{m \times m}$ is an orthogonal matrix, where the columns, known as the left singular vectors, form an orthonormal basis for the domain of $\mathbf{A}$.
- $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ is a rectangular diagonal matrix containing the singular values, which are non-negative quantities.
- $\mathbf{V} \in \mathbb{R}^{n \times n}$ is another orthogonal matrix, with its columns, referred to as the right singular vectors, providing an orthonormal basis for the codomain of $\mathbf{A}$.

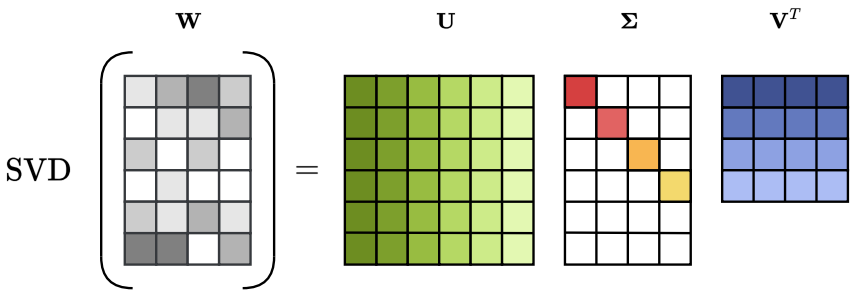


Figure 2.16: Illustration of the singular value decomposition.

Properties:

- **Existence**: SVD exists for all real matrices, whether square or rectangular, and it provides a complete description of their structure.
- **Ordering of singular values and vectors**: The singular values are conventionally arranged in descending order: $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$. The left and right singular vectors are similarly ordered to correspond with these singular values.
- **Singular values and rank**: The number of non-zero singular values is equal to the rank of $\mathbf{A}$, reflecting the count of linearly independent columns (and rows) in the matrix.

- **Expression as a sum of rank-1 matrices**: Through singular value decomposition, $\mathbf{A} \in \mathbb{R}^{m \times n}$ can be rewritten as a weighted sum of rank-one matrices:

$$\mathbf{A} = \sum_{i=1}^{\min(m,n)} \sigma_i u_i {v_i}^T,$$

  where:

  - r denotes the rank of $\mathbf{A}$,
  - u_i are the left singular vectors, the columns of $\mathbf{U}$,
  - v_i are the right singular vectors, the columns of $\mathbf{V}$,
  - σ_i are the singular values, serving as the weights in the decomposition.

Hadamard Decomposition

Hadamard decomposition represents a matrix as the element-wise, or Hadamard, product of two matrices. Given matrices $\mathbf{A} \in \mathbb{R}^{m \times n}$ and $\mathbf{B} \in \mathbb{R}^{m \times n}$, their Hadamard product is defined as:

$$\mathbf{A} \circ \mathbf{B} = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & \cdots & a_{1n} \cdot b_{1n} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} & \cdots & a_{2n} \cdot b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} \cdot b_{m1} & a_{m2} \cdot b_{m2} & \cdots & a_{mn} \cdot b_{mn} \end{bmatrix},$$

where each element of the resulting matrix is the product of the corresponding elements of $\mathbf{A}$ and $\mathbf{B}$.

3 | Related Works and State-of-the-Art Methods

This chapter explores the latest advancements in pruning and factorization techniques. As these approaches form the foundation of our study, a thorough review of recent developments offers essential context for the methodologies and analyses presented in the following chapters.

3.1. Pruning

Pruning is a well-established technique for reducing the size and complexity of neural networks. By eliminating redundant or less influential parameters, pruning seeks to construct a more efficient model while preserving its overall performance. The latest research in pruning methods for large language models explores both structured and unstructured approaches, each offering distinct advantages and trade-offs. These methods are examined in the following sections.

3.1.1. Structured Pruning

As detailed in Section 2.5.3, structured pruning involves the removal of predefined components within a model, such as neurons, channels, attention heads, or entire transformer blocks. This section reviews recent advancements in structured pruning, examining their methodologies, benefits, and inherent limitations.

LLM-Pruner [39] is designed for task-agnostic compression of large language models. The process begins by analyzing the model's structure to identify and group interdependent neurons, ensuring that entire functional clusters, rather than isolated parameters, are pruned together. To determine which groups to remove, LLM-Pruner applies a gradient-based importance estimation, assessing the impact of removing a structure on the loss function. This estimation leverages first-order gradients and an approximate Hessian matrix to rank groups by importance, selectively pruning the least critical ones. Following

pruning, the model undergoes a rapid recovery phase using LoRA, fine-tuning a small subset of parameters to efficiently restore performance.

SliceGPT [3] reduces the size of large language models by deleting entire rows or columns of weight matrices, effectively reducing the embedding dimension of the network. The method applies orthogonal transformations followed by their transposes in correspondence of the layer normalization modules, ensuring that the computations of the model remain unchanged. These transformations are computed based on PCA on the activations obtained from a calibration dataset, identifying the principal directions of the activation. Slicing is then performed by removing low-variance components, deleting corresponding rows and columns in the orthogonal matrices and their associated linear layers. This process preserves the most important features of the activations while minimizing approximation error. This simplification facilitates computational speed-ups on general-purpose accelerators.

Deja Vu, introduced by Liu et al. (2023) [37], is a structured pruning method designed to exploit contextual sparsity in large language models. It dynamically identifies subsets of attention heads and multi-layer perceptron neurons that are most relevant to a given input, enabling efficient pruning of the less relevant components without compromising the quality of the result. The method employs a lightweight predictor that determines essential components on-the-fly, leveraging contextual information from token embeddings at each layer. This method achieves over 2× speed-up during inference without requiring retraining. However, the prediction process introduces additional computational overhead, and the implementation is complex, often requiring hardware optimization. Challenges such as batching issues further limit its general applicability, despite its potential.

Other structured pruning techniques, such as SLEB, focus on pruning entire Transformer blocks to improve the efficiency of large language models. **SLEB**, developed by Song et al. (2024) [49], capitalizes on the observation that neighbouring Transformer blocks often produce highly similar outputs due to the incremental contributions each block makes to the residual path. This similarity indicates redundancy, which SLEB systematically identifies and removes.

Using a calibration dataset, SLEB evaluates the significance of each block by measuring its impact on token prediction results. It employs an iterative process where, after removing one block, the remaining blocks are re-evaluated to account for the changing importance of each component. This process ensures that only the least significant blocks are eliminated in subsequent steps. The method relies on a metric that assesses the reduction in log-likelihood when a block is removed, effectively quantifying its contribution

to the overall performance. This targeted pruning allows SLEB to maintain model quality while achieving substantial reductions in size and computational requirements. By removing entire blocks, SLEB avoids the sparsity issues associated with weight-level pruning, resulting in models that are more hardware-efficient and easier to deploy.

Early exit methods provide an alternative approach to structured pruning by determining whether additional layers should be processed based on the confidence of the model in its current output. Confidence, in this context, reflects the degree to which the current embeddings closely approximate the network's final output. When this similarity surpasses a predefined threshold, further computations are bypassed, significantly reducing inference time. While effective for tasks where confidence thresholds are well-defined, early exit methods often require substantial training to fine-tune the decision-making process and may face challenges in adapting to diverse or unpredictable input distributions.

3.1.2. Unstructured Pruning

As presented in Section 2.5.3, unstructured pruning operates at a fine-grained level by removing individual weights based on their significance. This section explores recent advances in unstructured pruning, examining methodologies, advantages, and limitations.

SparseGPT, introduced by Frantar and Alistarh (2023) [16], is an unstructured pruning method developed to efficiently reduce the size and computational requirements of large-scale language models. It achieves up to 60% sparsity with minimal degradation of accuracy by framing the pruning process as a series of sparse regression problems, solved independently for each layer. This layer-wise approach effectively identifies and removes parameters with negligible impact on the overall performance, enabling rapid and efficient compression.

The algorithm applies a lightweight optimization process, eliminating the need for global gradient updates or retraining. SparseGPT performs the pruning process in a single step, completing it efficiently even for models with an exceptionally large number of parameters. Despite its advantages, the irregular sparsity patterns generated by SparseGPT remain a challenge for hardware optimization, limiting its real-world deployment in some cases.

The **Wanda** method, proposed by Sun et al. [52], employs a refined unstructured pruning strategy that considers the interaction between weight magnitudes and input activations to guide the pruning process. Unlike conventional magnitude-based pruning, which evaluates weights in isolation, Wanda identifies and eliminates those weights with the least impact on the model output, enabling a more informed and accurate compression. This approach achieves significant sparsity while maintaining high performance, without

requiring retraining. By leveraging input-aware pruning criteria, Wanda effectively balances computational efficiency with the preservation of functional capabilities, offering a practical solution for optimizing large-scale language models.

3.2. Low-rank Factorization of Large Language Models

As introduced in Section 2.5.3, matrix factorization techniques have been explored to improve the efficiency of deep neural networks, particularly Transformer architectures, which are often constrained by their high dimensionality. The following sections review established techniques, key experiments, and promising approaches leveraging matrix factorization for model compression.

Truncated Singular Value Decomposition

Singular value decomposition can be utilized to obtain a low-rank approximation of a matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, a process referred to as truncated singular value decomposition. Truncated SVD approximates the original matrix by retaining only the largest singular values and their corresponding singular vectors. At rank k, the approximation is defined as:

$$\mathbf{A}_k = \mathbf{U}_k \cdot \mathbf{\Sigma}_k \cdot \mathbf{V}_k^T,$$

where:

- $\mathbf{U}_k \in \mathbb{R}^{m \times k}$ contains the first k columns of $\mathbf{U}$, corresponding to the k largest singular values, sorted in decreasing order.
- $\mathbf{\Sigma}_k \in \mathbb{R}^{k \times k}$ is a diagonal matrix with the largest k singular values,
- $\mathbf{V}_k \in \mathbb{R}^{n \times k}$ consists of the first k columns of $\mathbf{U}$, corresponding to the k largest singular values, sorted in decreasing order.

Properties:

- **Minimum approximation error**: Truncated singular value decomposition provides the best rank-k approximation of a matrix in terms of minimizing the Frobenius or spectral norm [13]. This ensures that the error in the low-rank approximation is the smallest possible when compared to other rank-k approximations.

Truncated SVD is a straightforward and effective method for approximating parameters of LLMs [42] [47]. It is suitable for compressing large weight matrices found in components

like attention mechanisms and feed-forward networks. Truncated SVD enables a low-rank approximation that retains the most critical information within the linear layers.

Truncated SVD offers the distinct advantage of minimizing approximation error, making it an ideal technique for low-rank approximations of isolated matrices. Its accessibility and straightforward implementation, facilitated by its integration into numerous computational libraries, further underscore its practicality for compressing models. However, this method comes with a significant drawback: the computational burden associated with processing very large matrices, as determining singular values and vectors is an inherently resource-intensive task.

DRONE

Introduced by Chen et al. [7], DRONE addresses a significant challenge in applying low-rank approximations to transformer-based models: the weight matrices in such models often exhibit a lack of inherent low-rank structure, resulting in high reconstruction errors when conventional methods like truncated SVD are used. By leveraging the observation that input activations typically reside in a lower-dimensional subspace, DRONE integrates the statistical properties of the input data distribution into the matrix decomposition process. This approach optimizes both the weight matrices and the input data distribution concurrently, enabling a more effective and data-aware compression framework.

The essence of DRONE lies in its mathematical formulation, where the low-rank approximation minimizes the reconstruction error of outputs rather than the weight matrix itself. This is achieved by incorporating the input data distribution during compression, ensuring that the approximation better aligns with the characteristics of real-world usage. Unlike traditional methods that only consider matrix properties, optimization in DRONE explicitly combines the input data and weight matrix properties, resulting in a provably optimal decomposition. A closed-form solution for this optimization problem guarantees efficiency in computation while preserving performance.

Experimental results underscore the capacity of DRONE to achieve substantial reductions in both model size and inference time. Notably, the approach also outperforms conventional SVD methods under equivalent constraints, as it more effectively captures the interplay between data and weight matrices.

Despite its advantages, DRONE assumes that the data distribution used during compression is representative of the deployment environment. If this assumption is violated, such as in cases with significantly shifted operational data, the method may yield suboptimal performance. Nevertheless, DRONE represents a notable advancement in low-rank

compression, offering a practical and theoretically grounded solution to reduce the computational and storage demands of large-scale NLP models.

Fisher-Weighted Singular Value Decomposition

Introduced by Hsu et al. [26], Fisher-Weighted Singular Value Decomposition (FWSVD) enhances standard low-rank factorization by incorporating Fisher information, a metric derived from the gradients of a loss function of a task, to assign importance to individual parameters. Unlike conventional SVD, which focuses solely on minimizing reconstruction error, FWSVD integrates task-specific information to ensure that compression efforts prioritize parameters critical to model performance. This approach addresses a common shortcoming of standard singular value decomposition, where indiscriminate compression may lead to a significant degradation in task accuracy.

Fisher-weighted singular value decomposition is particularly effective in fine-tuning task-specific language models, enabling significant compression while preserving task fidelity. The method is also versatile, as it can be applied to already compact models to achieve further reductions in size without compromising performance. By strategically preserving essential parameters identified through Fisher information, FWSVD minimizes the trade-off between compression and accuracy.

However, the approach introduces additional computational requirements due to the calculation of Fisher information, which involves evaluating parameter gradients relative to a loss function. This overhead may pose challenges for applications involving large datasets or scenarios demanding frequent retraining. Despite this limitation, FWSVD represents a compelling advancement in model compression, particularly for applications where task-specific accuracy is paramount.

Dense-Sparse Weight Factorization for Transformers (DSFormer)

Developed by Chand et al. [6], DSFormer introduces an innovative hybrid factorization scheme to address the shortcomings of traditional low-rank methods in compressing Transformer architectures. Recognizing that Transformer weight matrices often lack a strictly low-rank structure, DSFormer represents weights as a product of a small dense matrix and a semi-structured sparse matrix. This combination captures the underlying distribution of the parameters more effectively than low-rank approximations, yielding a superior balance between compression efficiency and task accuracy.

At the core of DSFormer lies the straight-through factorizer, a task-aware algorithm de-

signed to jointly optimize the dense and sparse components of the factorized weights. The straight-through factorizer overcomes the limitations of task-agnostic initialization by directly integrating task-specific objectives into the factorization process. This results in models that achieve higher compression ratios while maintaining competitive performance, even for demanding natural language understanding tasks.

Despite its effectiveness, the reliance of DSFormer on specialized factorization algorithms introduces additional implementation complexity. Furthermore, it requires careful adjustment of the dense-sparse structure to avoid compromising computational efficiency.

Activation-aware Singular Value Decomposition

Developed by Yuan et al. [55], Activation-aware Singular Value Decomposition (ASVD) introduces an approach to address key challenges in the low-rank decomposition of large language models. Unlike traditional SVD methods, which often overlook the impact of activation outliers, ASVD incorporates activation statistics into the decomposition process. By scaling weight matrices to align with activation distributions, ASVD minimizes reconstruction errors, particularly in layers sensitive to compression.

A central innovation of ASVD is its iterative calibration mechanism, which assigns varying compression ratios to layers based on their sensitivity to decomposition. This allows the method to adapt to the unique characteristics of each layer, optimizing the compression process while preserving model functionality. Additionally, ASVD extends beyond weight matrices to compress key-value caches, significantly reducing memory requirements without retraining the model.

Training-free design of ASVD ensures its practicality by leveraging a small calibration dataset to guide decompositions, avoiding the computational burden associated with retraining. While the method introduces additional complexity through scaling and iterative calibration, it provides a robust framework for reducing the computational and memory demands of modern NLP models while maintaining high accuracy.

SVD-LLM

Introduced by Wang et al. [54], SVD-LLM tackles key challenges in SVD-based compression of large language models by combining a truncation-aware data whitening strategy with a layer-wise closed-form parameter update mechanism.

The truncation-aware data whitening strategy transforms input activations by means of Cholesky decomposition, ensuring that individual channels are statistically independent

and have unit variance. This normalization simplifies the relationship between singular values and compression loss, allowing the method to predict the impact of truncating each singular value more effectively. By prioritizing the retention of singular values most critical to task accuracy, the strategy enables selective truncation that minimizes performance degradation, even at high compression ratios.

In parallel, the layer-wise closed-form parameter update iteratively refines the compressed weights after truncation. This mechanism aligns the weights with the updated activations of preceding layers, progressively reducing accuracy loss. The combination of these techniques provides a robust framework for achieving efficient model compression with minimal impact on task performance.

4 | Research Questions

This thesis explores redundancy in Transformer-based architectures, focusing on compression strategies designed to identify and streamline redundant or less significant components, thereby enhancing the understanding of their internal mechanisms and improving computational efficiency.

In this chapter, the discussion transitions to the central questions that underpin this investigation. These questions serve as a framework for examining redundancy and compression within Transformer architectures, shaping the research trajectory and providing the basis for the methods and results elaborated upon in the following chapters.

4.1. Research Question 1: How much and what redundancy exists in large language models?

Understanding the **extent of redundancy** within large language models is crucial to assess how much of their parameter space is essential for maintaining performance. By effectively quantifying redundancy, it becomes possible to identify pathways for model compression and optimization, facilitating the creation of smaller, more computationally efficient architectures that maintain comparable effectiveness. While extensive research has investigated this area, a comprehensive understanding of redundancy in Transformer architectures is still far from being reached.

A related avenue of inquiry concerns the specific **types of redundancy** inherent in these architectures. Given the hierarchical nature of Transformers, encompassing numerous modules and sub-modules, certain components may hold greater significance than others. Some modules could exhibit excessive repetition or lack meaningful contributions to overall performance.

Additionally, quantifying redundancy in such high-dimensional and interdependent parameter spaces presents substantial difficulties. A fundamental question is **how to systematically measure redundancy** within the parameter space. Redundancy may ap-

pear in complex patterns that are not immediately evident, making it challenging to define and isolate without compromising the operational integrity of the model. Thus, the task of characterizing and measuring redundancy in these complex parameter spaces remains significant challenge.

4.2. Research Question 2: Are the weight matrices in Transformer architectures inherently low-rank? If so, how can factorization techniques be applied to remove redundant or ineffective components in Transformers?

Weight matrices in linear layers encapsulate the core information within large language models, reflecting their learned capacity to understand and generate language. This research question delves into the structural characteristics of weight matrices in deep architectures, focusing on **whether matrices in Transformers exhibit inherent low-rank properties**. A low-rank matrix, in this context, is one where the majority of its variance or informational content can be represented using a smaller set of dimensions compared to its nominal size. Such a property suggests that, despite their high dimensionality, these matrices might contain significant redundancies or recurring patterns that enable a more compact representation.

Central to this inquiry is the **exploration of novel factorization techniques** as a means to identify and eliminate redundant or less effective components within weight matrices. An essential consideration is whether these matrices can be approximated in a low-rank form without incurring a substantial degradation in model performance. Addressing this question would contribute to a deeper understanding of the structural properties of Transformers while opening avenues for improved efficiency in their design and deployment. The insights gained could inform both theoretical exploration and practical advancements in optimizing large-scale language models.

4.3. Research Question 3: Are different weight matrices of the Transformer layers sharing similar features?

Transformer models consist of an extensive number of parameters, and it is plausible that some linear transformations may draw on analogous sub-transformations or recur across layers within the same block or between blocks in the architecture. This raises the question of **whether these weight matrices capture overlapping patterns or rely on shared features**.

A significant overlap in the informational content of a group of matrices would indicate the presence of dependent features. In contrast, the absence of shared features would suggest that the linear transformations carry out distinct computations and encode unique information.

Identifying shared features in high-dimensional weight spaces poses a substantial challenge. The complexity of these spaces makes it difficult to define or validate the shared components rigorously. Despite this, uncovering such features has profound implications for both understanding the architecture and optimizing it.

If repeated features are present, interest can be put on **whether they can be extracted and reused to design more efficient models by sharing parameters** across layers or creating generalized layers that encapsulate these common patterns. Beyond compression, uncovering shared features offers valuable insights into the internal mechanisms of Transformers and how knowledge is distributed across layers.

A key challenge lies in determining **how to isolate redundant computations within Transformer weights while ensuring the model retains the unique features necessary for layer-specific tasks**.

The central focus of this thesis is on matrix factorization as a method for compressing Transformer models by reducing redundancy in their weight matrices. This approach aims to achieve smaller, more efficient architectures without compromising their linguistic and reasoning capabilities. Building on low-rank approximation theory, the study further explores **whether matrix factorization can also be applied to exploit shared features across the model**. Specifically, this research examines **if factorization techniques can identify and represent these shared transformations in a compact form**, potentially enhancing efficiency while maintaining the functionality of deep architectures.

An essential question in this context is **whether compression through matrix factorization can be achieved independently or if additional fine-tuning is required** to restore or enhance performance. While fine-tuning might allow the model to adapt to its restructured form, it also introduces the risk of altering its representations in ways that limit its generalization ability or confine its functionality to the fine-tuning dataset.

The main challenge is to develop factorization techniques that strike a balance between reducing redundancy and preserving essential information. Additionally, the potential of combining parameter sharing with factorization remains largely unexplored, presenting a promising avenue for further investigation.

4.4. Research Question 4: Do modules in different blocks of the Transformers perform the same or similar functions?

In Transformer models, redundancies and useless computation can emerge not only within weight matrices but also at a higher structural level, such as modules or sub-modules. Determining **whether components like feed forward networks, self-attention mechanisms, or even entire Transformer blocks perform similar or overlapping functions across different parts of the architecture** is critical for identifying potential computational inefficiencies. If these components are found to exhibit functional similarities, it opens the possibility of simplifying the architecture by reducing or merging redundant operations, thereby creating more efficient models while maintaining comparable performance.

Uncovering these functional redundancies, however, is a complex task. The non-linear and input-dependent behavior of Transformer sub-modules, combined with the high dimensionality of their operations, makes distinguishing between similar and distinct functions particularly challenging. Addressing this question requires robust analytical frameworks and systematic reasoning about **how to compare and quantify functional overlap among modules**.

This research question further extends to investigating **whether certain layers or sub-modules in a Transformer can be combined to replace others, effectively reusing learned transformations to reduce the total number of unique components**. Such parameter sharing could lead to significant reductions in model size and computational requirements.

The primary challenge lies in ensuring that reusing or replacing layers does not disrupt the delicate balance of learned representations across the model. An aspect of this investigation is also **whether fine-tuning can address these disruptions**, potentially restoring performance and allowing the compressed model to adapt to its new configuration.

Ultimately, this research question aims to assess not only the extent of functional redundancy within Transformer models but also the practical methods for exploiting these redundancies to achieve efficient model compression.

4.5. Research Question 5: Are the corresponding modules at consecutive levels of the Transformer architecture more similar to each other in the sense that they have learned to perform the same or similar computation?

The sequential additive structure of Transformer models implies that consecutive layers are processing similar inputs and performing incremental transformations. This raises the question of **whether transformations performed by neighboring modules become progressively similar the closer they are in the stack**. Investigating this hypothesis can provide deeper insights into how Transformers process information across their architecture.

If significant similarity exists between consecutive layers, it could present opportunities for optimization. For instance, adjacent layers with similar transformations could be merged or share parameters, leading to a more compact and efficient model without compromising performance. This approach would shift the focus of compression from a global model-wide perspective to a more localized, adjacency-based strategy.

5 | Metrics, Datasets and Models

In this chapter, datasets, benchmarks, metrics, and models serving as tools for the analysis and experiments with Transformer-based architectures are presented.

By integrating these foundational resources, the chapter sets the stage for the evaluations and insights that are elaborated upon in the subsequent sections of the thesis.

5.1. Evaluation metrics

This section presents the evaluation metrics used to evaluate the architectures under study, covering measures of classification performance, language modeling quality, and compression efficiency. These metrics serve as essential tools for quantifying model effectiveness and enabling meaningful comparisons between different approaches.

Classification metrics

In binary classification tasks, the confusion matrix provides a comprehensive summary of classification outcomes, comparing predicted and actual class labels. Based on it, various evaluation measures are computed to assess model performance. Confusion matrix is structured as follows:

	Predicted Positive	Predicted Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

where:

- TP (True Positives) are the correctly predicted positive instances.
- TN (True Negatives) are the correctly predicted negative instances.
- FP (False Positives) are the incorrectly predicted positives (Type I error).
- FN (False Negatives) are the incorrectly predicted negatives (Type II error).

Using these values, several key evaluation metrics can be derived.

Accuracy

Accuracy measures the proportion of correct predictions relative to the total number of predictions:

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Accuracy offers a straightforward and general measure of performance. It can be misleading when applied to imbalanced datasets, where a class is significantly more frequent than the other.

Precision

Precision quantifies which fraction of the predicted positive instances are actually positive:

$$Prec = \frac{TP}{TP + FP} \tag{5.2}$$

A high precision means that when the model predicts a positive instance, it is likely correct. This metric is crucial in scenarios where false positives are costly.

Recall

Recall measures the proportion of actual positive instances that are correctly identified:

$$Rec = \frac{TP}{TP + FN} \tag{5.3}$$

A high recall ensures that most of the actual positive instances are correctly detected. This is particularly important in applications where missing a positive case (false negative) is costly.

F1-Score

The F1-score is the harmonic mean of precision and recall, providing a balanced measure when there is a trade-off between the two:

$$F1 = 2 \times \frac{Prec \times Rec}{Prec + Rec} \tag{5.4}$$

The F1-score is particularly useful when dealing with imbalanced datasets, as it considers both false positives and false negatives, unlike accuracy.

Perplexity

Perplexity is a statistical metric used to assess the predictive performance of a language model. It measures the uncertainty or confidence of the model in predicting the next word in a given sequence of words. A lower perplexity score signifies that the model assigns higher probabilities to the correct predictions, indicating superior language modeling capabilities. The mathematical definition is as follows:

$$PPL = 2^{-\frac{1}{N}\sum_{i=1}^{N}\log_2 P(w_i)},$$

where:

- N is total number of words in the sequence.
- $P(w_i)$ represents the probability assigned by the model to the i-th word.

Compression Ratio

The compression ratio quantifies the extent to which the size of a model is reduced following the application of a compression procedure. Lower ratios indicate more effective compression:

$$CR = \frac{\text{Compressed Model Size}}{\text{Original Model Size}}$$

5.2. Datasets

Datasets play a crucial role in natural language processing, serving as the foundation for training and evaluating language models. This section provides an overview of the datasets used in this study, detailing their composition and structure. Each dataset is introduced with a description of its size, linguistic diversity, and content.

5.2.1. WikiText-2

WikiText-2 [50] is a widely adopted dataset in natural language processing, commonly used for training and evaluating language models. Extracted from Wikipedia, it comprises curated and refined text passages, free from tables, lists, and other formatting artifacts, resulting in coherent, well-structured, and high-quality natural language sequences.

- **Size**: The dataset comprises 600 distinct articles extracted from Wikipedia, encompassing over 2 million tokens and approximately 33,000 unique tokens.
- **Diversity**: The dataset spans an extensive range of topics, including science, history, literature, culture, technology, and geography. Its content reflects the formal and factual writing style typical of Wikipedia, ensuring both structure and coherence.
- **Excerpt from the dataset**:

```
{
    "text": [...,
        " = = Development = = \n",
        "",
        " Concept work for Valkyria Chronicles III began
        after development finished on Valkyria Chronicles
        II in early 2010 , with full development beginning
        shortly after this . The director of Valkyria
        Chronicles II [...]",
        ...]
}
```

5.2.2. OpenWebText

OpenWebText [21] is a large-scale, open-source dataset created to replicate the high-quality corpus used in training OpenAI's GPT-2 model. It draws its content from URLs that were highly upvoted on Reddit, ensuring that the text represents diverse, engaging, and high-quality content from across the web.

- **Size**: Approximately 40 GB of uncompressed text data, containing billions of tokens.
- **Diversity**: The dataset covers a wide variety of topics, including science, technology, culture, and everyday human experiences, making it an excellent resource for

training and evaluating general-purpose language models. It includes both structured and unstructured text formats, such as articles, discussions, essays, and creative writing.

- **Excerpt from the dataset**:

```
{
    "text": [...,
        "Ad blockers are often painted as the enemy of
        online publishers, but sometimes things are more
        complicated.\n\nAdBlock Plus, for example, just
        announced that t h e y re working with startup
        [...]",
        ...]
}
```

5.3. Benchmarks

Benchmarks provide standardized tasks that test different aspects of model performance, from common-sense reasoning to complex problem solving. They are used in the following chapters for they robustness and their ease of understanding evaluation results.

5.3.1. TruthfulQA

TruthfulQA [36] is a benchmark designed to evaluate the truthfulness and factual accuracy of language models. It consists of a set of carefully crafted questions that are intended to elicit incorrect or misleading responses from the models. The primary goal is to assess how well a language model can avoid generating false or fabricated information, especially when faced with tricky prompts.

- **Task**: The goal is to answer multiple choices of questions. In this task, a given sentence is sequentially combined with each possible continuation from the set of choices. The model assigns a probability score to each resulting sentence. The continuation that produces the sentence with the highest probability score is selected as the response of the model.
- **Metric**: The metric used to evaluate the LLM on the task is the accuracy.
- **Size**: The benchmark comprises 817 questions, each developed to rigorously evalu-

ate the ability of LLMs to distinguish between truthful and fabricated information.

- **Diversity**: The questions span 38 categories, including topics such as health, law, finance, and common misconceptions, providing a wide range of scenarios to challenge factual accuracy.

5.3.2. HellaSwag

HellaSwag [56] serves as a benchmark dataset specifically designed to assess the capabilities of language models in commonsense reasoning and natural language understanding. It comprises a collection of questions, each presenting an incomplete narrative or scenario. The model has to identify the most plausible continuation.

- **Task**: This benchmark employs a multiple-choice sentence continuation task. Similarly to TruthfulQA, each possible choice is appended to the scenario described in the question to form a complete sentence. These sentences are then evaluated on the basis of their generation probabilities under the model, and the choice that achieves the highest probability score is selected as the predicted answer.
- **Metric**: Accuracy is the metric used to evaluate performance on this benchmark, measuring the proportion of correct predictions.
- **Size**: HellaSwag consists of 70000 questions, making it a sizeable dataset for benchmarking large language models.
- **Diversity**: The dataset covers a broad spectrum of topics, including everyday scenarios, physical activities, procedural knowledge, abstract reasoning tasks and common sense knowledge. Its diversity ensures a comprehensive evaluation of LLM capabilities in reasoning and understanding across various contexts.

5.3.3. GSM8K

GSM8K [9] is a benchmark consisting of meticulously curated grade school math word problems. It is designed to assess the mathematical reasoning and problem-solving skills of language models. The problems are presented in natural language and span a broad array of topics typically included in elementary and middle school curricula. Successfully solving GSM8K problems requires models to demonstrate a deep understanding of mathematical principles and reasoning, as well as the ability to parse and resolve word problems articulated in natural language.

- **Task**: The dataset focuses on solving free-text mathematical problems. Models are

tasked with generating a coherent and accurate solution to the problem described in the input text.

- **Metric**: Performance is evaluated using exact match accuracy, which measures the proportion of answers that perfectly match the correct solution.
- **Size**: The benchmark comprises 8500 problems.
- **Diversity**: GSM8K covers a wide variety of mathematical topics, including arithmetic, algebra, geometry, and word-based reasoning problems. This diversity ensures that evaluations are reflective of general problem-solving and reasoning capabilities of the model across multiple mathematical domains.

5.4. Models

This section presents the models that will be used in the subsequent analysis. Each model will be introduced with a description of its architecture and key characteristics. This overview will provide the necessary context for the evaluations and comparisons conducted in later chapters.

The experiments utilized three categories of neural network architectures: Mistral, Llama, and Gemma. Specifically, the following pre-trained models available on Huggingface were employed:

- **Llama-3.1-8B**
- **Mistral-7B-v0.3**
- **gemma-2-2b**
- **gemma-2-9b**

5.4.1. Llama Models

- *Model ID on HuggingFace:* ***meta-llama/Llama-3.1-8B***

Llama 3.1 8B [12] is part of the Llama series by Meta AI. It is a decoder-only Transformer model with 8 billion parameters, tailored to efficiently address a wide range of natural language processing tasks, including text generation and long-context applications.

This model employs a Transformer architecture comprising 32 blocks. Each block incorporates self-attention layers and feed forward networks, adhering to the main foundational principles of the original Transformer framework.

Llama-3.1-8B integrates grouped-query attention and key-value caching mechanisms. As explained in Section 2.4.2, GQA streamlines the computational demands of multi-head attention by grouping query heads and using the same key and value heads for the elements in a group, while key-value caching allows the reuse of key and value representations from preceding tokens during inference. These integrations significantly improve efficiency and minimize latency, particularly in tasks that involve extended or interactive sequences.

The normalization strategy chosen by the architecture is root-mean-squared layer normalization [57], which scales activations using the root-mean-square of the input without the need for mean subtraction. This approach reduces computational complexity while maintaining training stability.

Tokens are embedded in a 4096-dimensional space, enabling the model to effectively capture the semantics of words. The vocabulary consists of 128 256 tokens, supporting enhanced performance in multilingual and domain-specific tasks. Rotary positional embeddings (Section 2.4.2), a fundamental component of the design, facilitate effective positional encoding across input sequences without imposing additional computational demands.

The model supports a maximum context length of 128 000 tokens, making it well-suited for applications requiring extensive context comprehension.

Llama 3.1 was pre-trained on approximately 15 trillion tokens of publicly available text, primarily sourced from the Web, using a next-token prediction objective. Data cleaning strategies were rigorously applied to ensure high-quality inputs, with a data mix consisting of approximately 50% general knowledge, 25% mathematical and reasoning tasks, 17% programming-related content, and 8% multilingual tokens.

5.4.2. Mistral Models

- *Model ID on HuggingFace:* ***mistralai/Mistral-7B-v0.3***

The Mistral models, developed by Mistral AI, represent an innovative family of machine learning architectures designed to redefine efficiency and performance in natural language processing. Mistral AI, a leading organization in AI research, specializes in creating high-performance, open-weight large language models with a focus on resource efficiency.

Mistral-7B-v0.3 [28], a member of this family, is a decoder-only Transformer-based large language model consisting of 7.3 billion parameters. It is optimized for generative tasks in NLP, with its autoregressive design aligning with the requirements of text generation.

The architecture of Mistral-7B-v0.3 includes 32 Transformer blocks, combining multi-head self-attention with feed forward networks. It employs grouped-query attention to optimize the performance of multi-head attention, reducing memory and computational overheads in large-scale applications. Key-value caching further enhances inference efficiency, particularly in sequential or extended data processing.

The initial version, Mistral-0.1, introduced GQA in conjunction with a sliding window attention mechanism. This mechanism reduced the computational overhead of standard attention by constraining its scope to a fixed-sized window around each token, enabling faster processing of long sequences by focusing on local context. However, the sliding window mechanism was removed in later iterations.

Root-mean-squared layer normalization is utilized to improve training stability and computational efficiency by providing consistent scaling of activations.

Mistral-7B-v0.3 uses embeddings with a dimensionality of 4096. The vocabulary used by Mistral-7B-v0.3 is composed by 32,768 tokens. Rotary positional embeddings, initially introduced in version 0.1 and preserved in subsequent versions, enhance the capacity of the model to encode positional information effectively.

Version 0.2 expanded the context length of the architecture from 8,192 tokens in version 0.1 to 32,768 tokens, allowing improved handling of long sequences. This advancement remains in version 0.3.

Although the specifics of the training dataset have not been disclosed, the model is trained on a diverse corpus of publicly available online text, ensuring broad and versatile comprehension of natural language.

5.4.3. Gemma Models

- *Model ID on HuggingFace for 2-billion variant:* ***google/gemma-2-2b***
- *Model ID on HuggingFace for 9-billion variant:* ***google/gemma-2-9b***

Gemma is a series of state-of-the-art, lightweight open-weight models developed collaboratively by Google DeepMind and other research teams at Google. These models build on the technological foundation established by the Gemini series [2].

Gemma-2-2B [46], with 2.6 billion parameters, balances compactness and efficiency, delivering robust performance for diverse tasks. In contrast, Gemma-2-9B [46], featuring 9.2 billion parameters, is equipped to handle more complex linguistic scenarios, demonstrating enhanced capabilities in text understanding and generation.

Both models utilize a decoder-only Transformer architecture and integrate cutting-edge techniques. The 2.6-billion variant consists of 26 Transformer blocks, while the 9.2-billion version comprises 42 blocks. Grouped-query attention optimizes the multi-head attention mechanism, reducing computational costs without sacrificing quality. Alternating layers of local sliding window attention (4096 tokens) and global attention (8192 tokens) further improve efficiency.

Key-value caching accelerates inference for extended sequences by reusing prior computations. Root-mean-squared layer normalization, applied in both pre- and post-layer normalization, ensures consistent input and output scaling, stabilizing training.

The embedding dimensionality differs across models, with 2.6 billion parameters using 2304-dimensional embeddings and the 9.2 billion variant employing 3584-dimensional representations, capturing complex semantic relationships. The models have expanded vocabularies of 256 000 tokens. RoPE method is employed as the positional encoding mechanism, effectively enhancing the capacity to handle long-context applications.

The Gemma-2 models support context lengths of up to 8192 tokens.

The pre-training phase utilized a corpus of publicly available text, predominantly in English, sourced from web documents, code repositories, and scientific articles. Both variants employed knowledge distillation, with the 27B model serving as teacher. The 9B model was trained on approximately 8 trillion tokens, while the 2B model utilized a corpus of around 2 trillion tokens.

Following pre-training, a post-training pipeline was implemented, utilizing techniques such as supervised fine-tuning and reinforcement learning from human feedback. This approach further enhanced the performance of the models, ensuring their ability to address complex and diverse tasks effectively.

6 | Compression Methods

This chapter introduces the theoretical foundation of the novel compression approaches developed in this thesis to reduce the size of LLMs. Building on concepts of parameter redundancy and shared features, which will be showed and discussed in detail in the next chapter, it introduces strategies that leverage these redundancies.

6.1. Matrix Aggregation and Sharing Strategy (MASS)

The MASS aims to leverage shared features between linear layers to achieve effective compression. By aggregating multiple components into a single shared representation, the objective of MASS is to reduce the parameter count while preserving the core functionality of the model.

6.1.1. MASS Method

The proposed method encompasses two primary phases: the grouping phase and the aggregation phase.

1. **Grouping Phase**. Layers selected for compression are divided into subsets using a grouping strategy that guarantees structural compatibility and ensures redundancy or similarity between layers.
2. **Aggregation Phase**. Within each subset, weight matrices and biases are merged into a single shared representations using a chosen aggregation strategy that defines how the information is combined.

6.1.2. Grouping

Matrices in the transformer model take on various roles, such as query, key, value, and other transformations and are distributed across different transformer blocks. To create shared layers, a grouping criterion must be defined.

A key requirement is that all matrices within a group must have the same shape to ensure

compatibility during the aggregation process.

Choosing an appropriate grouping strategy is important, as matrices that share similar features lead to more effective compression, resulting in smaller approximation errors and better overall performance.

Within this thesis, groups are determined based on matrix type. Additionally, experiments are conducted without applying compression and matrix sharing to the first and last layers, which have distinct roles, as highlighted by the findings in Section 7.1.3.

Finding the best grouping criterion is an interesting research topic left for future work.

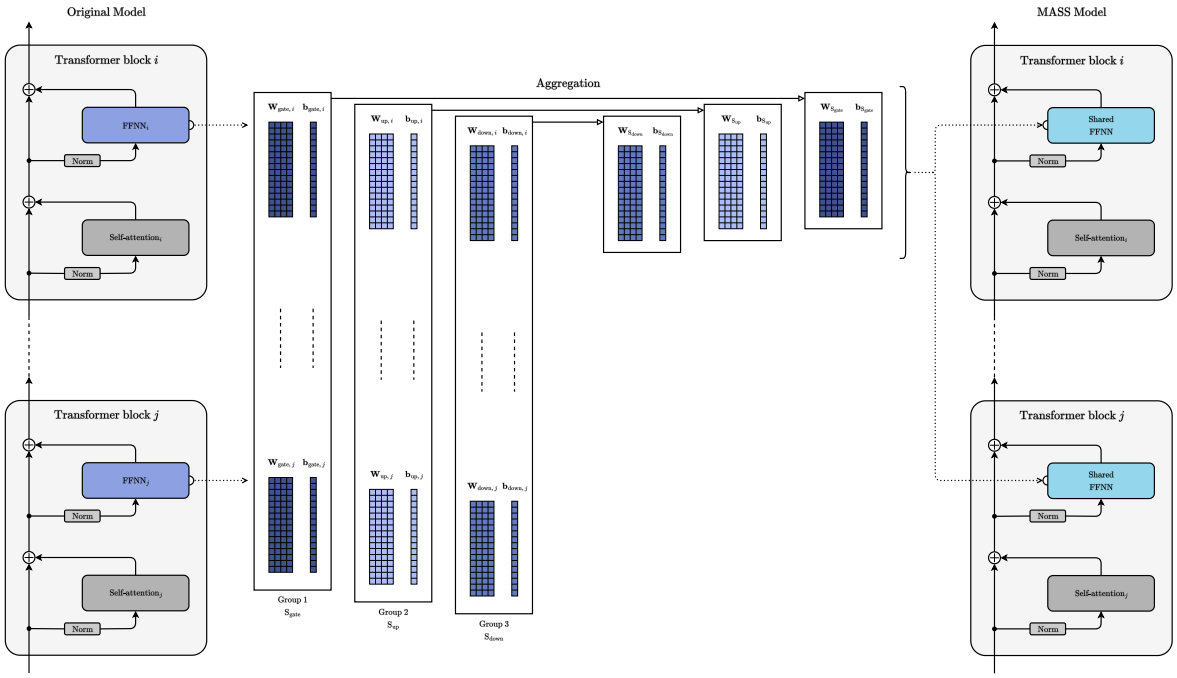


Figure 6.1: Illustration of the MASS method applied to the matrices of FFNN modules, grouped by layer type.

6.1.3. Aggregation Criterion

In this study, the MASS methodology adopts simple averaging as the aggregation criterion to compute shared layers.

For a given group S_i, consisting of the weight matrices and biases, if present, of N_i linear layers from a deep model, $S_i = \{(\mathbf{W}_1, \mathbf{b}_1), (\mathbf{W}_2, \mathbf{b}_2), \ldots, (\mathbf{W}_{N_i}, \mathbf{b}_{N_i})\}$, the parameters of the aggregated layer are computed by averaging the weights and biases of the original

layers:

$$\mathbf{W}_{\mathrm{S}_i} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{W}_j, \tag{6.1}$$

$$\mathbf{b}_{\mathrm{S}_i} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{b}_j. \tag{6.2}$$

Here, $\mathbf{W}_{\mathrm{S}_i}$ and $\mathbf{b}_{\mathrm{S}_i}$ denote respectively the weight and bias parameters of the resulting shared layer.

As discussed in Section 6.1.1, different aggregation strategies enforce the layers to share specific aspects or characteristics. The choice of an appropriate aggregation strategy should be informed by the nature of the features that are intended to be shared.

The simple averaging strategy operates under the assumption that the weight matrices of linear layers exhibit a comparable structural alignment. For this approach to be effective, the corresponding elements in these matrices must not only hold a similar semantic meaning but also be consistent in their magnitude and positional arrangement. This consistency is crucial to ensure that the averaged elements yield a matrix that faithfully encapsulates the shared characteristics of the original matrices, rather than resulting in a representation that is incoherent.

Although this study focuses on simple averaging, other aggregation strategies, such as weighted averaging or clustering-based methods, might provide superior results by more effectively capturing the shared features of the layers. Exploring such alternatives is suggested as a direction for future research.

Fine-Tuning

When layers are replaced with a shared representation, the original learned features and capabilities of the model might be affected due to the loss of layer-specific nuances. Fine-tuning offers a solution to this challenge by retraining the modified model on task-specific or general training data. This process enables the model to adjust its parameters and mitigate the impact of changes introduced by MASS.

6.2. Global and Local Factorization (GlobaL Fact)

The GlobaL Fact method introduces a systematic framework to compress transformer architectures by decomposing selected linear layers into two low-rank components. One

component is shared across multiple layers, while the other remains unique to each module. This dual representation seeks to balance the trade-off between parameter sharing and the retention of layer-specific functionalities, enabling reduced inter-layer redundancy while preserving model performance.

6.2.1. GlobaL Fact Method

The GlobaL Fact methodology comprises two main stages: grouping and factorization. Additionally, if required, a retraining phase can be introduced to address and reduce the impact of approximation error.

1. **Grouping.** This stage organizes the layers targeted for compression into subsets, denoted as $\mathrm{S}_1, \mathrm{S}_2, \ldots, \mathrm{S}_K,$. Central to this phase is the adoption of a grouping strategy, which determines the layers that shares features and should be based on their structural compatibility and similarity.

2. **Factorization.** In the factorization stage, each matrix is decomposed to achieve a compact representation. Let N matrices $\mathbf{W}_j \in \mathbb{R}^{m \times n}$, derived from linear layers in a transformer model, be grouped in S_i. Their approximations, denoted as $\tilde{\mathbf{W}}_j \in \mathbb{R}^{m \times n}$, are represented using a shared matrix $\mathbf{G}_{\mathrm{S}_i} \in \mathbb{R}^{r \times n}$ and unique local matrices $\mathbf{L}_j \in \mathbb{R}^{m \times r}$. The factorized representations are expressed as:

$$\begin{aligned} \tilde{\mathbf{W}}_1 &= \mathbf{L}_1 \cdot \mathbf{G}_{\mathrm{S}_i}, \\ \tilde{\mathbf{W}}_2 &= \mathbf{L}_2 \cdot \mathbf{G}_{\mathrm{S}_i}, \\ &\vdots \\ \tilde{\mathbf{W}}_N &= \mathbf{L}_N \cdot \mathbf{G}_{\mathrm{S}_i}. \end{aligned} \tag{6.3}$$

   The internal dimension of the factorization is denoted as r. It commonly referred to as the rank because it directly relates to the rank of the approximation. In particular when r is significantly smaller than the dimensions of the original matrix, as is often the case in compression, the resulting matrix product will approximately have a rank of r. More precisely, the rank of the approximated matrix is upper-bounded by r, provided that $r < m$ and $r < n$.

   For an input vector $\mathbf{x} \in \mathbb{R}^n$, the linear transformation $\mathbf{W}_j \in \mathbb{R}^{m \times n}$ is applied as:

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{W}_j^T.$$

Considering equations 6.3, an input $\mathbf{x}$ is first processed by the global matrix and subsequently by the local matrix.

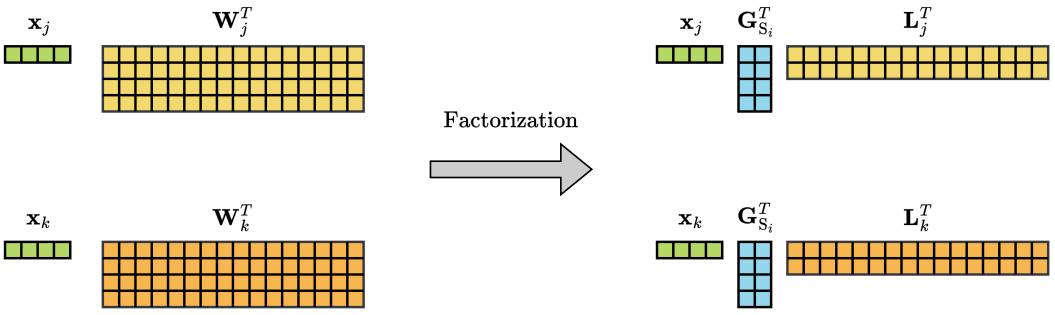


Figure 6.2: Illustration of the GlobaL Fact framework applied to the matrices $\mathbf{W}_j$ and $\mathbf{W}_k$. The vector $\mathbf{x}_j$ is the input to the linear transformation $\mathbf{W}_j$, while $\mathbf{x}_k$ is the input to $\mathbf{W}_k$.

An alternative configuration of GlobaL Fact involves reversing this order of the factors, where the input is first processed by the local matrix and then the global matrix:

$$\begin{aligned} \tilde{\mathbf{W}}_1 &= \mathbf{G}_{S_i} \cdot \mathbf{L}_1, \\ \tilde{\mathbf{W}}_2 &= \mathbf{G}_{S_i} \cdot \mathbf{L}_2, \\ &\vdots \\ \tilde{\mathbf{W}}_N &= \mathbf{G}_{S_i} \cdot \mathbf{L}_N. \end{aligned} \tag{6.4}$$

Here, $\mathbf{G}_{S_i} \in \mathbb{R}^{m \times r}$ and $\mathbf{L}_j \in \mathbb{R}^{r \times n}$.

This study explores only the first configuration; therefore, subsequent discussions will focus exclusively on this case.

The matrices $\mathbf{G}_{S_i}$ and $\mathbf{L}_j$ are the two core components of this framework:

- **Global Matrices**($\mathbf{G}_{S_i}$): Shared across multiple layers, these matrices encapsulate features recurring throughout the architecture. Multiple global matrices can be employed, based on the grouping strategy, allowing for structural refinement, with each group linked to a distinct global matrix.
- **Local Matrices ($\mathbf{L}_j$)**: Specific to each layer, these matrices retain the operational details unique to individual layers.

The GlobaL Fact technique is founded on the observation that, while Transformer layers perform diverse functions, certain features may recur or align across layers to some degree. By capturing these shared features in the global matrix, this approach aims to extend the

capabilities of low-rank approximations to achieve greater compression. Meanwhile, the local matrix safeguards the unique characteristics of individual layers.

6.2.2. Grouping

As explained in Section 6.2.1, applying factorization to compress the model requires defining a grouping strategy for the layers.

To enable shared transformations, an essential assumption is that all matrices in a group must either have the same shape or, as minimum requirement, be compatible with the shape of the shared global matrix.

Selecting an appropriate grouping strategy is crucial, as the more features matrices in the same group share, the more effective the factorization will be. Effective grouping results in smaller approximation errors and better overall performance.

In this thesis, grouping is determined based on layer type.

Finding the best grouping criterion is an interesting research topic left for future work.

6.2.3. Factorization

Given a set of weight matrices, deriving the local and global components required for factorization is a complex yet critical step. While SVD and its variants, discussed in Section 2.5.4, are well-suited for decomposing and possibly approximating non-shared matrices, the shared nature of global matrices across multiple layers introduces additional challenges for their direct application. Nevertheless, various approaches are available to initialize both global and local matrices. This section explores these alternatives.

A straightforward approach involves initializing all matrices randomly and subsequently optimizing them with respect to the original model's target matrices. Although viable, this method often converges slowly and fails to leverage potential similarities across layers during the global matrix construction. To address these limitations, alternative initialization strategies that exploit the properties of the layers are considered.

Initialization of the Global Matrices

The initialization of the global matrices is designed to extract shared components from the weights of the grouped layers.

1. **Initialization via Right Singular Vectors of the Average Matrix.** The global matrix is initialized by selecting the top r right singular vectors, ranked according

to their corresponding singular values, of the sum of the matrices within a group S_i. This approach relies on the assumption that the matrices in the group share similar weight distributions, making their aggregated sum a suitable representation for identifying shared components:

$$\begin{aligned} \mathbf{U}_{S_i},\, \mathbf{\Sigma}_{S_i},\, \mathbf{V}_{S_i}^T &= \text{SVD}\left(\sum_{\mathbf{W}_j \in S_i} \mathbf{W}_j\right), \\ \mathbf{G}_{S_i} &= \mathbf{V}_{S_i;r}^T, \end{aligned} \tag{6.5}$$

where $\mathbf{V}_{S_i;r}^T$ represents the right singular vectors associated with the top r singular values of the average matrix of group S_i.

2. **Initialization via Average of Right Singular Vectors.** This initialization sets the global matrix to the average of the matrices composed by the most relevant right singular vectors, obtained from the SVD of each layer in the group S_i. This approach operates on the assumption that layers within a group share similar principal directions on average:

$$\begin{aligned} \mathbf{U}_{S_i,j},\, \mathbf{\Sigma}_{S_i,j},\, \mathbf{V}_{S_i,j}^T &= \text{SVD}(\mathbf{W}_j), \\ \mathbf{G}_{S_i} &= \sum_{j=1}^{|S_i|} \mathbf{V}_{S_i,j;r}^T, \end{aligned} \tag{6.6}$$

where $\mathbf{V}_{S_i,j;r}^T$ represents the right singular vectors associated with the top r singular values of each matrix $\mathbf{W}_j$ within the group S_i.

3. **Clustering of Singular Vectors** The first initialization strategy may result in reduced similarity between layers, as it does not account for variations in the weights distribution and in the arrangement of internal dimensions. The second method maps singular vectors of different matrices one-to-one based on their significance within their respective matrices. This approach may fail to capture deeper similarities; for example, the most significant singular vector in one matrix might align more closely with the second most significant singular vector in another. Such mismatches can lead to less accurate similarity measures and suboptimal initialization. A more effective approach could involve comparing singular vectors based on their directions rather than relying solely on their importance within a matrix.

   To address this limitation, an alternative approach involves clustering (obtaining r distinct sets) the right singular vectors of matrices within the same group. The

centroids of the r clusters, or other representative elements selected through an alternative method, are then used as the basis for initializing the global matrix. This strategy has the potential to improve the similarity and alignment of shared features across layers, resulting in a more robust and effective initialization process.

The clustering-based initialization method has not been implemented and tested in this work. Its practical application and evaluation are left for future research.

Although these methods obtain low approximation errors, their computational cost, particularly for large-scale language models, may be significant due to the SVD computations required.

The initialization time required for method 2 is comparable to the time needed to compute the SVD approximation of the layers targeted for compression, making it similar to the baseline represented by truncated SVD. Conversely, method 1 is significantly faster, as it requires computing the SVD for only a single matrix rather than for each matrix in the group. Method 3, on the other hand, would be the most computationally expensive, as it combines the initialization time of method 2 with the additional time needed for clustering. The latter varies depending on the chosen clustering strategy.

Initialization of the Local Matrices

The initialization of the local matrices $\mathbf{L}_j$ is achieved by projecting the original matrices $\mathbf{W}_j$, from group S_i, onto the pseudo-inverse of the global matrix $\mathbf{G}_{\mathrm{S}_i}$:

$$\mathbf{L}_j = \mathbf{W}_j \cdot \text{pseudo-inverse}\left(\mathbf{G}_{\mathrm{S}_i}\right), \quad j = 1, 2, \ldots, N. \tag{6.7}$$

This approach ensures that the local matrices are aligned with both the original weights and the global matrix during initialization. Although this initialization is suboptimal, it provides a structured starting point for subsequent optimization.

Initialization Pre-Training

To refine the initialization, a brief pre-training phase can be performed. During this stage, the initialized matrices are optimized relative to the target matrices from the original model, reducing approximation errors and improving alignment with the target model.

Fine-Tuning

Replacing layers with shared representations may alter the original features and capabilities learned by the model due to the approximations introduced by GlobaL Fact. Fine-tuning mitigates this issue by retraining the modified model on task-specific or general training data, enabling it to adjust its parameters and compensate for the introduced changes.

The Global-Local factorization approach, built on principles distinct from those of non-shared factorization, aims to assess the feasibility and effectiveness of shared matrices in leveraging redundant components to achieve enhanced outcomes. In this context, fine-tuning serves as a mechanism to refine the model further, ensuring that the global features successfully capture and share meaningful knowledge, including aspects that may not have been adequately identified during initialization.

6.3. Adapter-Based Approximation and Compression Optimization (ABACO)

Compression methods and, in particular factorization-based ones, such as singular value decomposition or Global-Local factorization (discussed in Section 6.2), may potentially face challenges arising from their discontinuous nature. These methods often truncate matrices abruptly, removing large portions of the parameters in one step. Although effective in reducing model size, such approaches frequently degrade performance, requiring extensive fine-tuning to recover lost capabilities. Moreover, this truncation may fail to account for the nuanced interplay between parameters, particularly in architectures as interconnected as transformers.

To address these limitations, ABACO provides an iterative and smoother strategy. Instead of directly removing parameters, ABACO method gradually shifts the functionality encoded in the pre-trained weights of the model into a set of low-rank matrices. This approach leverages adapter-based fine-tuning techniques, incorporating a penalization term that progressively diminishes the contribution of the original weights, thereby encouraging the transfer of knowledge from the full model into the low-rank components.

The theory underlying ABACO, as described here, primarily utilizes LoRA adapters introduced in Section 2.4.3, though its principles are applicable to other adapter-based methods as well.

ABACO Process

1. **Adapter Integration.** The first step in ABACO involves introducing modified LoRA adapters on top of the original weights of the pre-trained model. This method employs an adaptive scaling parameter, α, to balance the contributions of the pre-trained weights and the adapters effectively. Given the original weight matrix $\mathbf{W}_0 \in \mathbb{R}^{m \times n}$ of the model and the trainable adapter weights, $\mathbf{A} \in \mathbb{R}^{m \times r}$ and $\mathbf{B} \in \mathbb{R}^{r \times n}$, the adapted resulting matrix is represented as:

$$\mathbf{W} = \alpha \mathbf{W}_0 + \mathbf{A} \cdot \mathbf{B}.$$

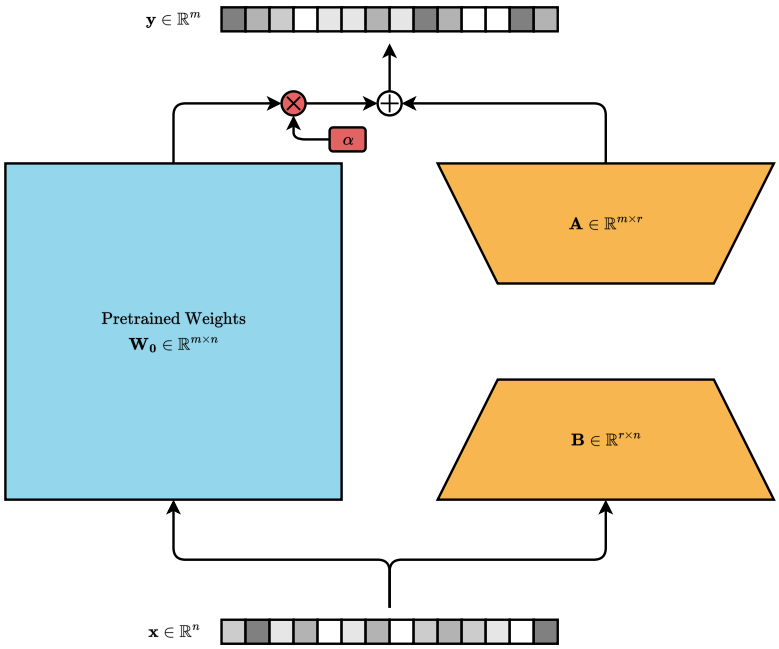


Figure 6.3: Diagram of ABACO applied to the weight matrix $\mathbf{W}_0$.

In the ABACO framework, the low-rank matrices are trained while the frozen original pre-trained weights are simultaneously penalized through the scaling of α.

2. **Iterative Compression.** During training, α is gradually reduced following a decay schedule, decreasing the reliance on the pre-trained weights while emphasizing the role of the adapters. This gradual transition should allow the model to adapt to its compressed state and retain performance.

   The ABACO framework leverages fine-tuning as the central mechanism for transferring knowledge from the original model to the low-rank adapters. The selection

of fine-tuning data plays a decisive role in shaping the resulting compressed model. Opting for a more general dataset ensures the retention of broad language and reasoning capabilities of the model, keeping it suitable for general-purpose applications. Conversely, choosing a specialized dataset leads to a more focused model tailored to specific tasks or domains, aligning its functionality with specialized objectives.

3. **Pre-trained weights discarding or approximation.** At a certain stage of training, pre-trained weights are either entirely discarded or replaced with an approximation.

This approach enables compression through a continuous process, allowing the model to adapt iteratively while preserving performance and gradually reducing redundancy. This smoother transition could help prevent the pitfalls of sudden parameter removal.

Penalization Scaling

During the optimization of the low-rank adapters, the method gradually reduces α over time, balancing the diminishing reliance on the original weights with preserving performance. Several decay strategies can be employed:

1. **Linear decay**

   Linear decay provides a constant reduction of α over a predefined number of iterations or epochs:

   $$\alpha = \max\left(0, \alpha_0 \cdot \left(1 - \frac{t}{T_{\text{horizon}}}\right)\right),$$

   - α_0 is the initial value assumed by α.
   - t represents the current iteration of the fine-tuning procedure.
   - T_{horizon} is the total number of iterations over which α decays to 0.

2. **Exponential Decay**

   A smoother transition in the final stages of training can be achieved using exponential scaling:

   $$\alpha = \alpha_0 \cdot b^{-c \cdot t}$$

   - α_0 is the initial value assumed by α.
   - b is the base of the exponential decay, commonly chosen as 2, e, or 10.
   - t represents the current iteration of the fine-tuning procedure.
   - c controls the decay rate of α.

Careful selection of the hyperparameters defining the scaling strategy is essential to ensure a smooth transition, preventing performance degradation and enabling effective compression.

Practical Implications of ABACO

ABACO builds on existing adapter-based approaches, such as LoRA, requiring only minimal changes to their implementation. This allows for seamless adoption without introducing complex preprocessing steps or manipulations.

The method is thought to be used in resource-constrained deployment environments. The approach can be suited in specific settings and domains with constrained resources both at training and inference time, with a focus on a small inference latency.

Although low-rank adapters are inherently suitable for settings with constrained training resources, they still require the full model during inference. ABACO leverages the training efficiency of low-rank adapters and extends their applicability by generating lightweight models that are optimized for deployment on devices with limited memory and computational capacity.

7 | Experiments and Results

This chapter delves into the experimental investigations conducted to assess and exploit redundancies within transformer-based large language models. These experiments aim to bridge theoretical insights into redundancy with practical model compression techniques, ensuring a balance between reducing computational demands and maintaining performance.

The research is structured around two synergistic phases. The first phase focuses on identifying redundancies and irrelevant features at various levels of the transformer architecture, from individual layers to higher-level modules. Building on these insights, the second phase explores compression strategies presented in Chapter 6. Although these phases are presented separately for clarity, they are intrinsically interdependent. Insights from redundancy analysis directly inform the development of compression methodologies, while the outcomes of these compression experiments, in turn, validate and refine the understanding of redundancy.

Key findings are discussed, supported by both quantitative metrics and qualitative insights, highlighting the interplay between architectural complexity, compression strategies, and model performance.

7.1. Experiments on Redundancy

This section explores the concept of redundancy and inefficiencies within Transformer-based architectures, detailing the experiments conducted to evaluate these aspects. Results of the analyses are presented and critically discussed to provide insights into the potential for optimization and enhancement.

7.1.1. Redundancy Analysis

Deep models, and in particular LLMs, are becoming every day more powerful and more general solving tasks that until some year ago were responsibility of man only. In addition to new state-of-art architectures and methods, this improvement happens together with

and is often attributed to the growth in the number of model parameters.

This escalation to billions and even trillions of weights have raised questions about the usefulness of this parameters or at least about whether the same job can be dove with a smaller number of them.

The lack of explainability in deep neural architectures and their training processes limits the possibility to clearly discern the role of each component, understanding the functions they perform, and analyzing how these functions are learned. This inherent opacity complicates the identification of redundant or superfluous elements within the structure. In particular, for large language models, it remains unclear what knowledge the system acquires, what it merely memorizes, and how this information is applied in practice.

Human language, while immensely rich and diverse, has a finite nature. It conforms to a bounded set of grammatical and semantic rules, demonstrating significant repetition both in the sentences that can be constructed and in the core concepts that can be conveyed. The overall universe of documents is similarly constrained, with a particularly finite subset of information available for training purposes. Remarkably, even human cognition, operating with a finite number of neurons and synapses, achieves extraordinary levels of sophistication, extending far beyond fundamental tasks like reading and writing.

Consequently, one might expect that capturing the essence of language does not require infinite or excessively large parameter spaces. Although large-scale language models typically gain performance with additional parameters, research increasingly suggests that many of these parameters could encode repetitive or marginally useful information. Thus, the premise that "more parameters necessarily lead to better understanding" deserves scrutiny. Instead, more efficient model architectures and training methods may prove just as powerful than their massively scaled counterparts, preserving the language capabilities of Transformers without their unsustainable demand for computational resources.

In the context of Transformer, this thesis refers to redundancy as the presence of repeated or noisy components that could be removed without causing significant damage to the performance of the model.

In order to address the research question 1 (Section 4.1), a classification of redundancy types in large language models has been carried out.

1. **Intra-Layer Redundancy**: Individual weight matrices may contain irrelevant components or capture noise from the training data. There are ways to reduce this redundancy by removing the noisy elements from matrices obtaining more compact forms.

2. **Inter-Layer Redundancy**: Layers may compute similar functions or partially overlapping operations. For example, in the case of linear layers, redundancy can be detected by assessing the similarity between their weight matrices and bias vectors.
3. **Inter-Module Redundancy**: Redundancy arising between higher-level modules composed of multiple layers.
    - **Inter-Sub-Block Redundancy**: Sub-modules, such as the self-attention mechanism or feed forward neural network layers, can sometimes perform overlapping or redundant functions relative to other sub-modules in the model.
    - **Inter-Block Redundancy**: Entire blocks within the transformer stack may apply similar transformations to their inputs, leading to unnecessary repetition in their functionality.

The amount of redundancy inside a model can vary based on some factors:

- **Domain-Awareness**: Since weights are often trained on large, diverse datasets, certain weights may become redundant or irrelevant for specific domains. Applying a general LLMs in a limited context can lead to portions of its knowledge remaining unused.
- **Task-Awareness**: The type of task also influences redundancy in the model, as some tasks require more complex processing than others.

7.1.2. Rank Analysis

A fundamental objective of the thesis is to explore model compression through factorization techniques. To evaluate the viability of low-rank approximation, This analysis examine the extent to which the matrices involved in linear transformations within Transformer architectures exhibit low-rank characteristics, both individually and collectively. In particular, the presence of low-rank structures can be investigated at multiple levels: by analyzing individual weight matrices (intra-layer redundancy) or by exploring interdependencies across matrices from different layers (inter-layer redundancy).

Singular Values Distribution and Approximate Rank

Weight matrices are analyzed by evaluating the distribution of their singular values and employing a scalar measure referred to as the approximate rank.

Singular values, obtained via SVD (Section 2.5.4), provide a measure of the distribution of information content encoded within a matrix. Considering the SVD decomposition in

its formulation as a sum of unitary rank matrices, the distribution of singular values offers insights into the contribution of each component to the overall structure. By analyzing these values, it is possible to discern whether specific singular values, along with their corresponding singular vectors, play a dominant role.

From the singular values, cumulative explained variance is computed. Given a matrix $\mathbf{X} \in \mathbb{R}^{m \times n}$ having singular values σ_i, the cumulative explained variance when considering the first k singular values is defined as:

$$EV_k = \frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{\min(n,m)} \sigma_i^2}. \tag{7.1}$$

This metric quantifies the proportion of the total variance retained when preserving only the k most significant components.

The explained variance provides a measure of approximation error when reconstructing the matrix using only the top k singular components (truncated SVD, Section 3.2). If a small subset of k singular values accounts for a high cumulative explained variance, approximating the original matrix using only the top k components would yield a low approximation error.

The rank of a matrix reflects the degree of correlation among its rows and columns, indicating the number of independent directions required to represent the transformation. A high level of correlation among elements suggests significant compressibility, reinforcing the feasibility of low-rank approximations. The rank of a matrix corresponds to the number of its non-zero singular values.

Within Transformer blocks, matrices are generally not inherently low-rank, meaning they rarely contain singular values that are precisely zero. However, certain dimensions may contribute minimally to the overall information content. In the context of these experiments, the approximate rank is defined as the number of singular values required to reach a predetermined threshold, such as 80%, 90% or 95%, of the cumulative explained variance. This represents the number of components necessary to retain the majority of the variance in the data.

Intra-Layers Rank Analysis

For each linear layer within the self-attention and feed forward modules, redundancy is assessed through the following steps:

1. **Decomposition of the Weight Matrix**: singular value decomposition is applied to the weight matrices extracted from the model.
2. **Cumulative Explained Variance Computation**.
3. **Approximate Rank Computation by Thresholding**.

The analysis is conducted on Llama 3.1-8B (Section 5.4.1), Mistral-7B (Section 5.4.2), and Gemma-2B and Gemma-9B (Section 5.4.3).

This section presents the obtained plots and compares them across models. The plots for the Llama model are shown here, while the corresponding diagrams for the other models are provided in Appendix A, Section A.1.

Examining the figures reveals that directly comparing the magnitudes of singular values is inherently challenging. These values vary depending on the model and layer type, although they generally remain within a constrained range across the four models analyzed. Notably, the Llama model exhibits several matrices with singular values significantly larger than those observed in other models and layer types. Differences in magnitude, however, lack a straightforward interpretation.

Despite these variations in absolute value, certain coherent patterns emerge. In relation to research questions 1 (4.1) and 2 (4.2), several layers display characteristics indicative of an approximate low-rank structure. Although the extent varies by layer type, information is highly concentrated in the dominant singular components. These observations are supported by Figure 7.8, which demonstrates that a limited number of singular values are sufficient to capture 90% of the variance.

These findings suggest that dimensionality reduction through low-rank approximations is a viable approach to approximate layers without substantially affecting model performance, reinforcing the validity of factorization-based compression techniques previously explored in the literature (Section 3.2).

When considering layer types, query and key matrices tend to exhibit lower ranks, particularly in the Llama and Mistral models. Similarly, self-attention output matrices, responsible for merging results from attention heads, also display low-rank characteristics, albeit less pronounced than those observed in query matrices. In contrast, value matrices tend to show a greater number of significant singular values, resulting in a higher

rank. This increased rank may stem from their crucial role in information propagation, as value weight matrices are the primary contributors to the computation of subsequent embeddings.

Feed forward layers generally exhibit less pronounced low-rank behavior, reflecting a greater retention of significant features. This trend may be attributed to their larger matrix dimensions, typically $h \times 4h$ or $4h \times h$, which inherently involve more parameters and contribute to higher rank values.

Up-projection transformations, in particular, consistently exhibit higher ranks across all models. During training, the weights of these layers within FFNN blocks are optimized to enhance feature diversity within the expanded $4h$-dimensional space. This optimization likely explains the observed higher-rank behavior, as the model leverages a greater number of components to effectively transform and encode information.

Although matrices in Gemma 2 models require a higher number of components for accurate approximations, the considerations outlined in this section remain applicable to all the analyzed models.

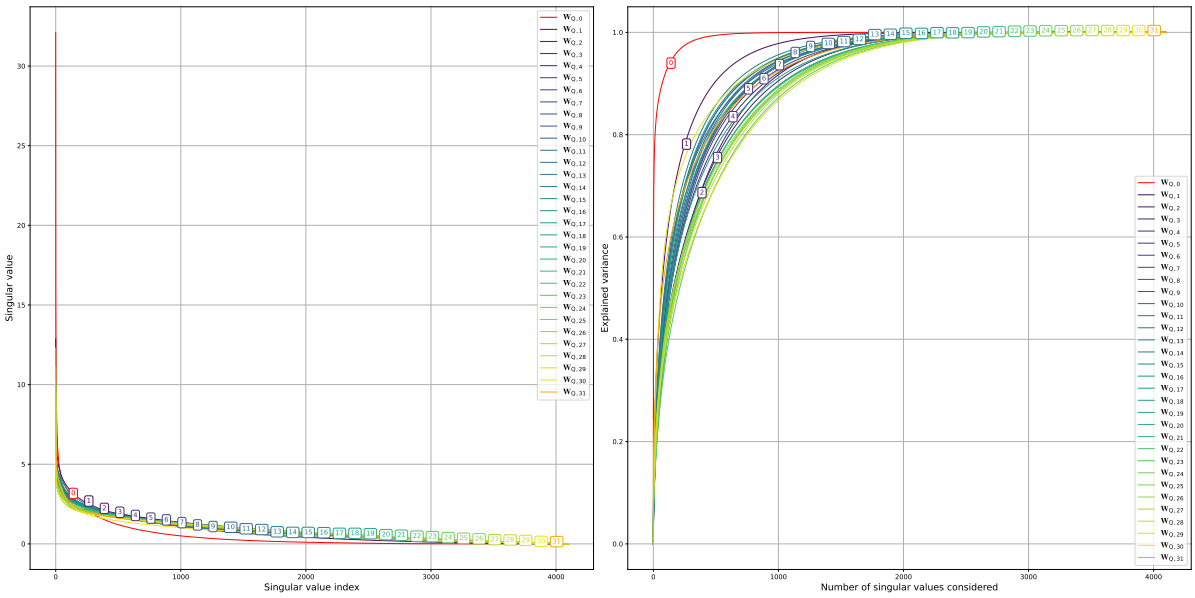


Figure 7.1: Singular value distribution and cumulative explained variance of self-attention query matrices in Llama 3.1 blocks.

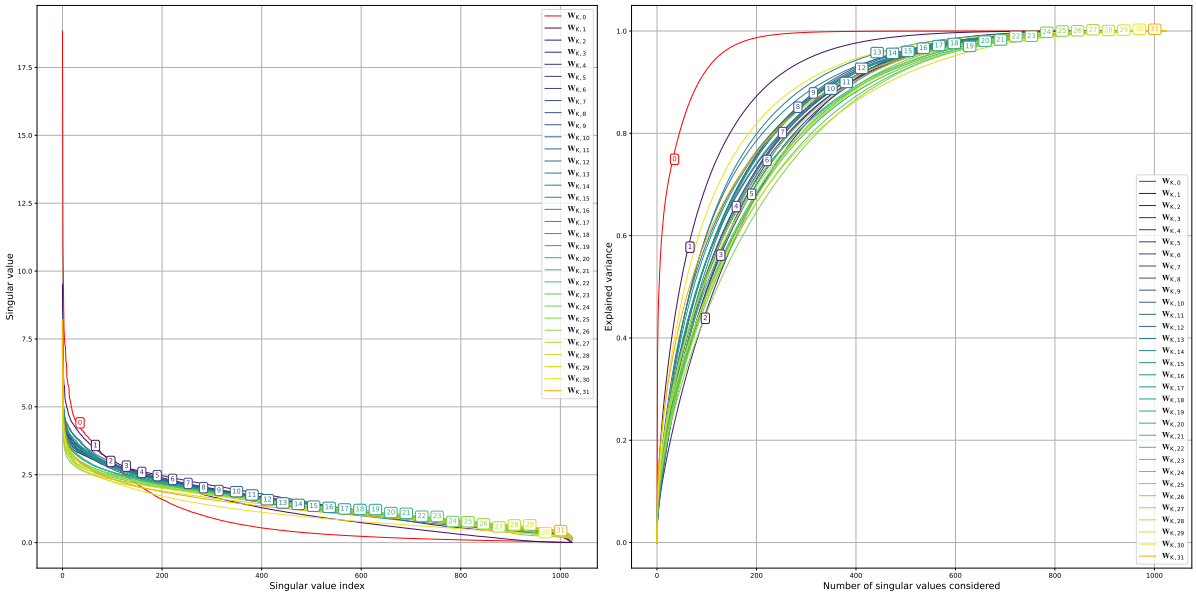


Figure 7.2: Singular value distribution and cumulative explained variance of self-attention key matrices in Llama 3.1 blocks.

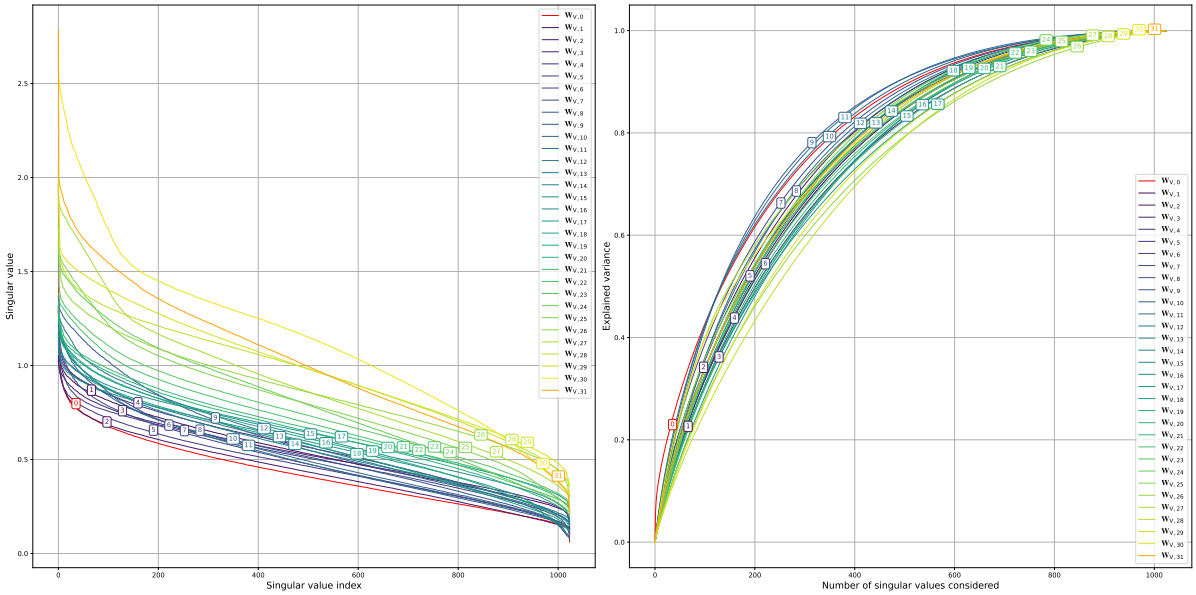


Figure 7.3: Singular value distribution and cumulative explained variance of self-attention value matrices in Llama 3.1 blocks.

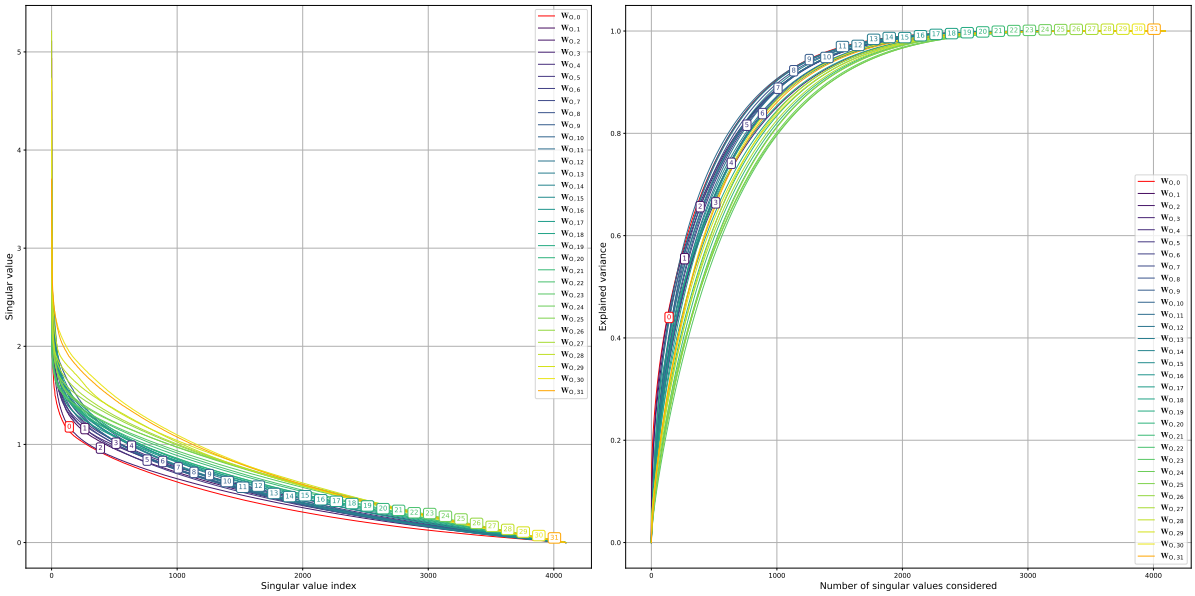


Figure 7.4: Singular value distribution and cumulative explained variance of self-attention output matrices in Llama 3.1 blocks.

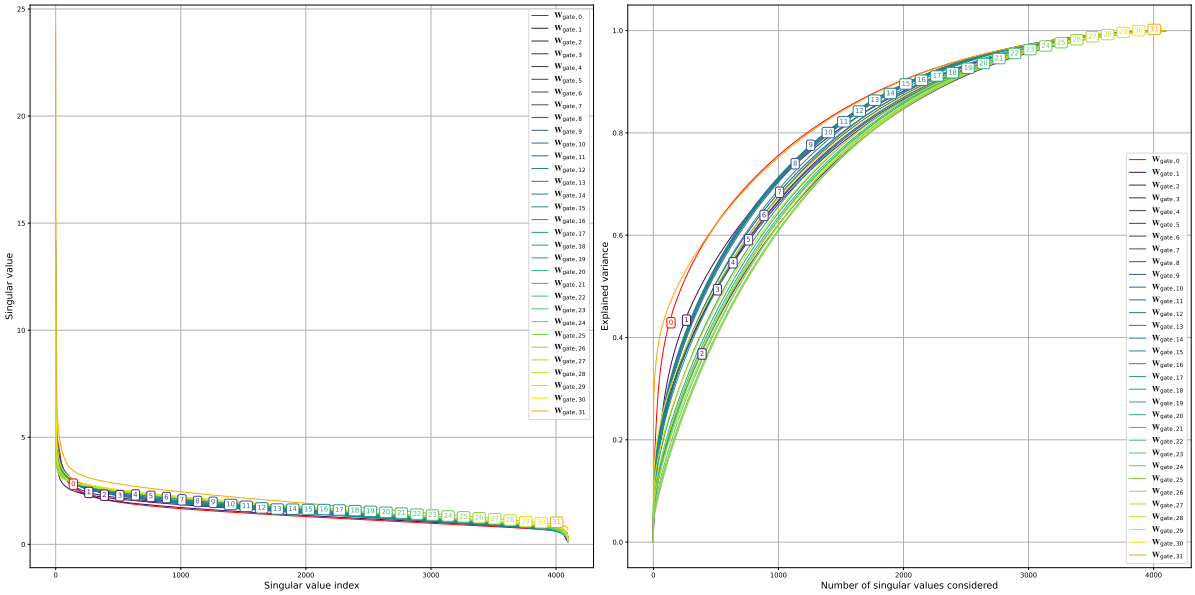


Figure 7.5: Singular value distribution and cumulative explained variance of gate projection matrices in Llama 3.1 blocks.

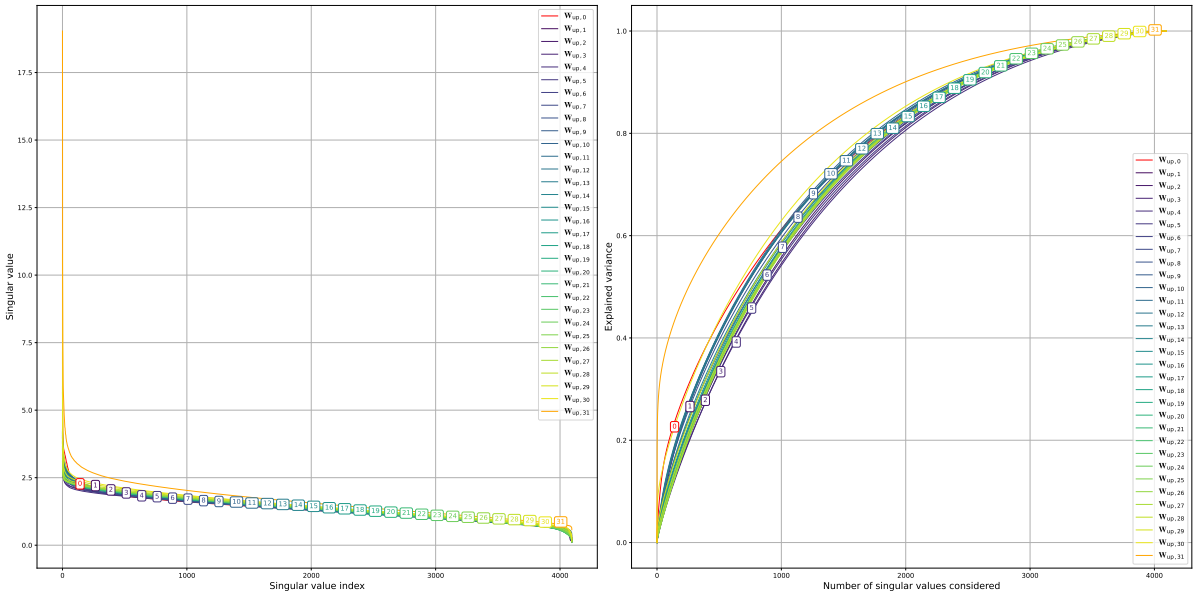


Figure 7.6: Singular value distribution and cumulative explained variance of up projection matrices in Llama 3.1 blocks.

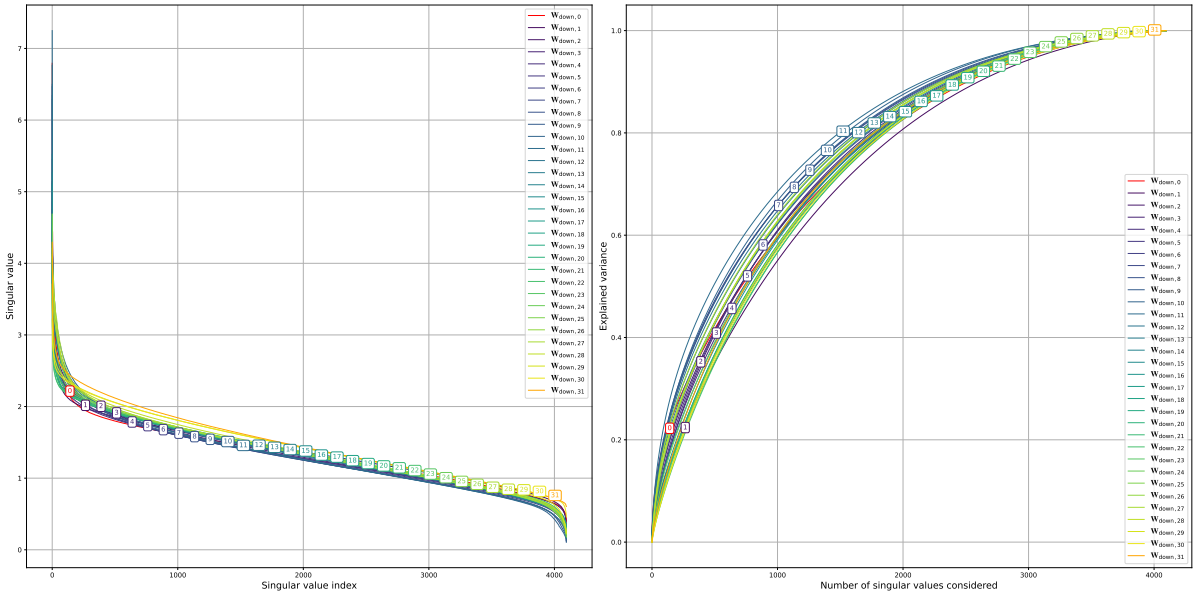


Figure 7.7: Singular value distribution and cumulative explained variance of down projection matrices in Llama 3.1 blocks.

Figure 7.8: Approximate ranks of the weight matrices in Llama 3.1, determined using a threshold of 0.9 for the explained variance.

Inter-Layers Rank Analysis

Another aspect under analysis is whether layers in Transformer architectures rely on similar underlying computations, as discussed in Research Question 4.3. Identifying shared global features across different layers requires more than analyzing individual matrices in isolation. Matrices of linear layers are organized into groups, and the relationships within each group are analyzed. To assess inter-layers redundancy, rank analysis is performed on the concatenation of matrices within a group of layers. This method provides insight into the correlation between columns and rows of different transformations. If strong correlations emerge among weight matrices from different layers, resulting in a low-rank concatenated matrix, it may indicate redundant computations, suggesting potential opportunities for optimization.

Rank Analysis of Concatenated Layers

Redundancy across multiple layers is analyzed by performing the following steps:

1. **Concatenation of the Weight Matrices**: Weight matrices extracted from the model are grouped and concatenated based on their layer type. Specifically, matrices with the same role (e.g., query, key, etc.) from different blocks are combined. Other grouping criteria are possible and left for future work.

   Within each group, two concatenation strategies can be employed:

   (a) Row-Wise Concatenation: Matrices are stacked vertically, preserving the number of columns, which corresponds to the input dimensionality.

   (b) Column-Wise Concatenation: Matrices are stacked horizontally, preserving the

number of rows, which corresponds to the output dimensionality.

2. **Decomposition of the Weight Matrix**: Singular value decomposition is applied to the result of the concatenation.
3. **Cumulative Explained Variance Computation**.
4. **Approximate Rank Computation by Thresholding**.

The analysis is conducted on Llama 3.1-8B (Section 5.4.1), Mistral-7B (Section 5.4.2), and Gemma-2B.

This section presents the obtained plots and compares them across models. The plots for the Llama model are shown here, while the corresponding diagrams for the other models are provided in Appendix A, Section A.2.

Upon concatenation, the singular value distributions exhibit many relevant singular values of comparable magnitude, indicating that redundancy across layers is not excessively pronounced. If the weight matrices were highly correlated, at least one of the two concatenation strategies would show a steep decline in singular values and a rapid increase in cumulative explained variance, suggesting a strongly low-rank structure. Instead, the distributions remain broad, implying that different layers introduce sufficient variation to maintain a high rank.

Notably, the elevated ranks in Figures 7.23 and 7.24 correspond to a number of parameters that is higher for concatenated matrices than the sum of those needed when layers are considered separately. This suggests that concatenation does not introduce substantial exploitable redundancy but instead increases structural complexity.

Rank estimates differ between row-wise and column-wise concatenation for gate, up, down, key, and value projection matrices due to the mismatch between their input and output dimensions. In contrast, query and self-attention output matrices are square, resulting in concatenated matrices of identical shape for both concatenation strategies. Interestingly, row-wise staking appears to concentrate more information in the dominant singular components of those layers.

Overall, the findings indicate that while matrices do not collapse into a strongly low-rank structure. This suggests that Transformer blocks contribute distinct transformations, reinforcing the idea that each layer captures unique aspects of the learned representations. Consequently, exploiting redundancy for optimization may be more challenging than anticipated.

Similar observations apply when analyzing the plots and approximated rank values for

the other models contained in Section A.2.

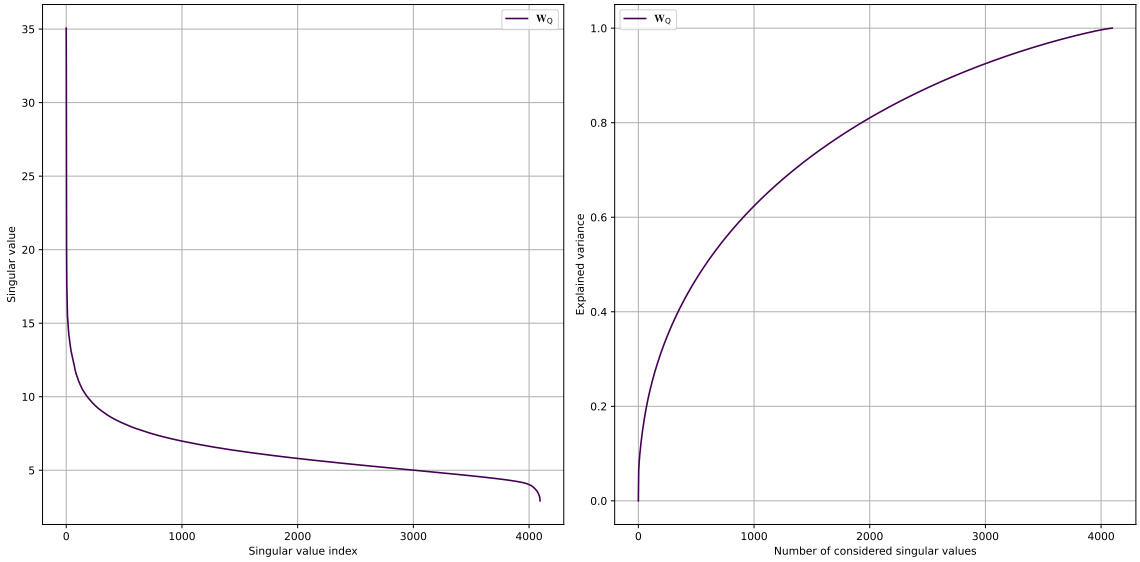


Figure 7.9: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention query matrices in Llama 3.1 blocks.

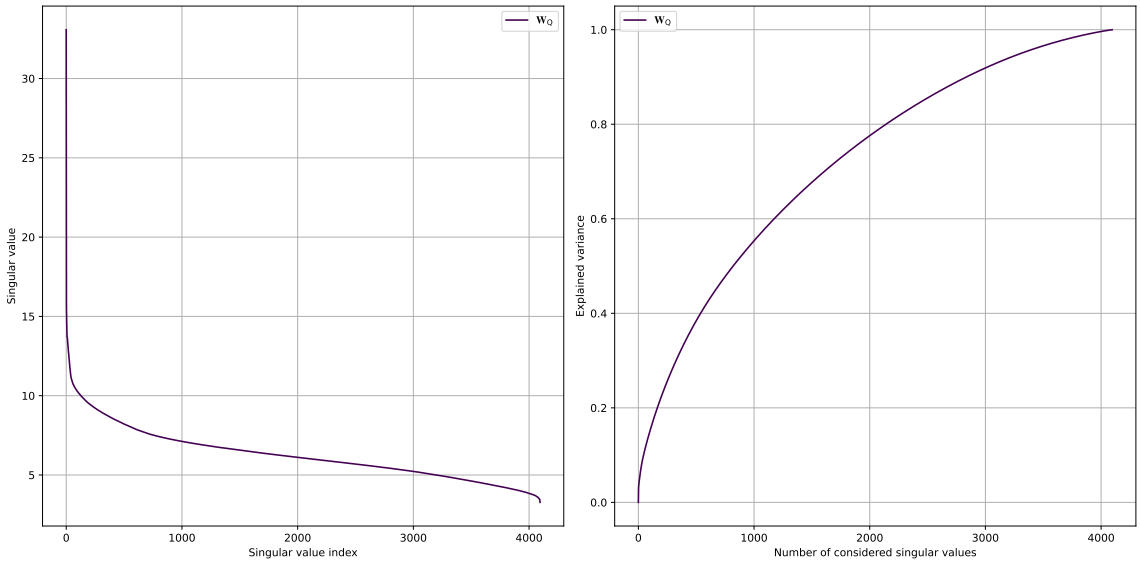


Figure 7.10: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention query matrices in Llama 3.1 blocks.

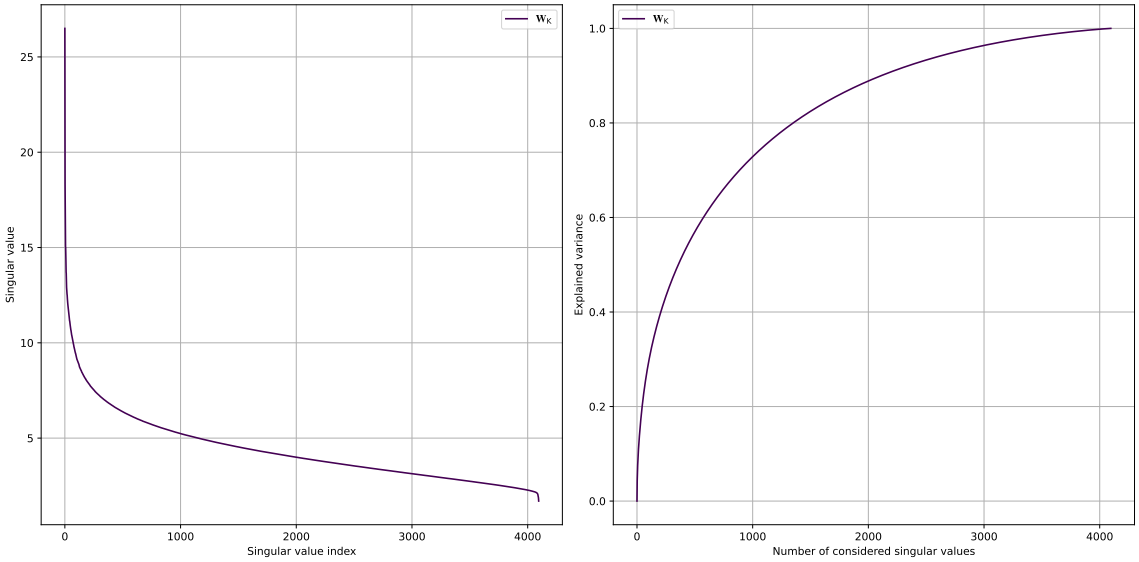


Figure 7.11: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention key matrices in Llama 3.1 blocks.

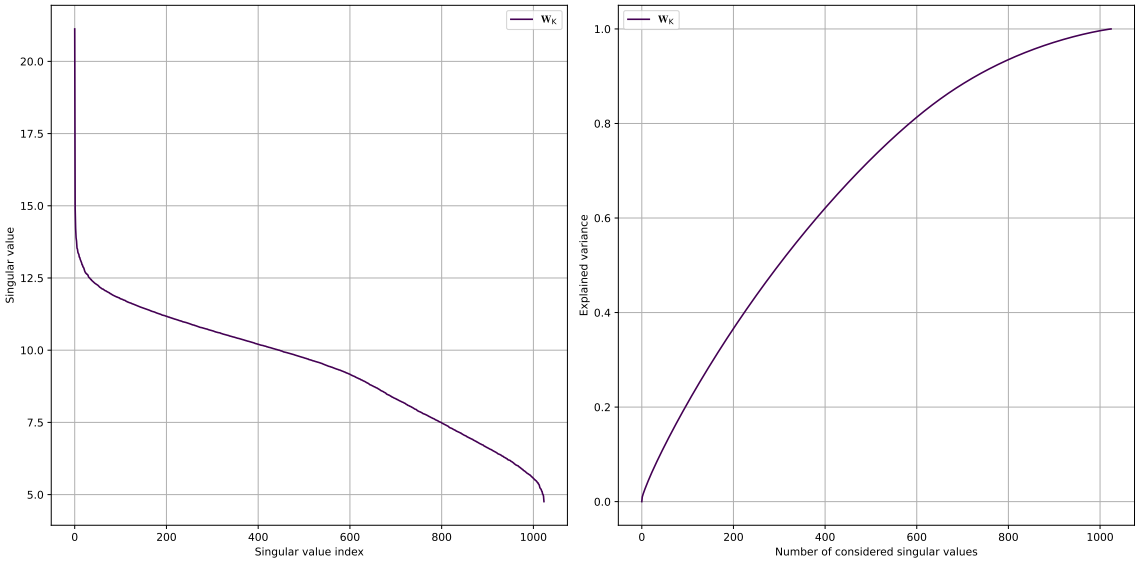


Figure 7.12: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention key matrices in Llama 3.1 blocks.

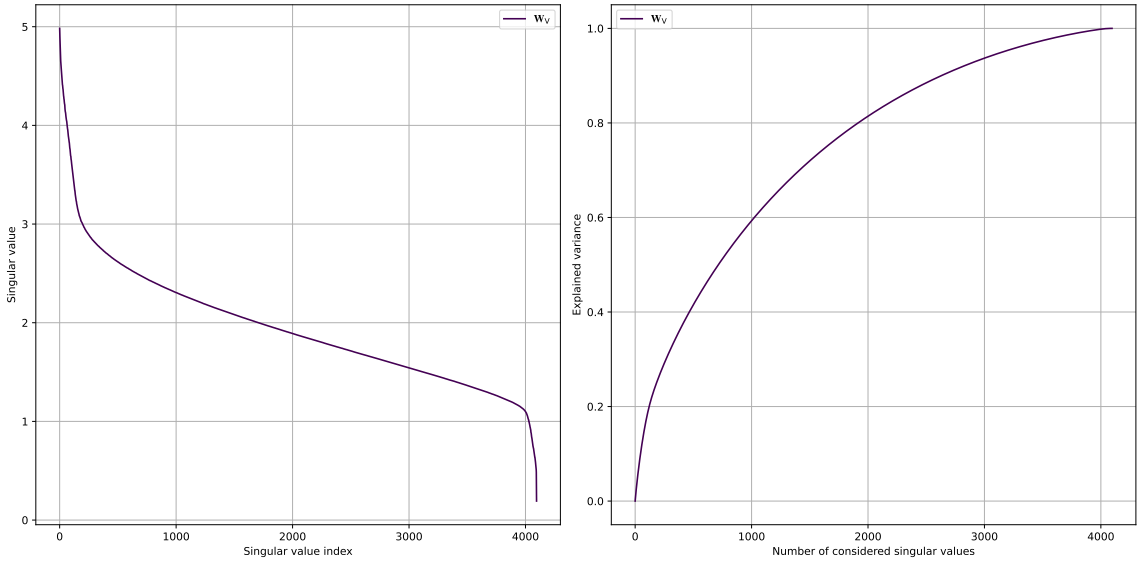


Figure 7.13: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention value matrices in Llama 3.1 blocks.

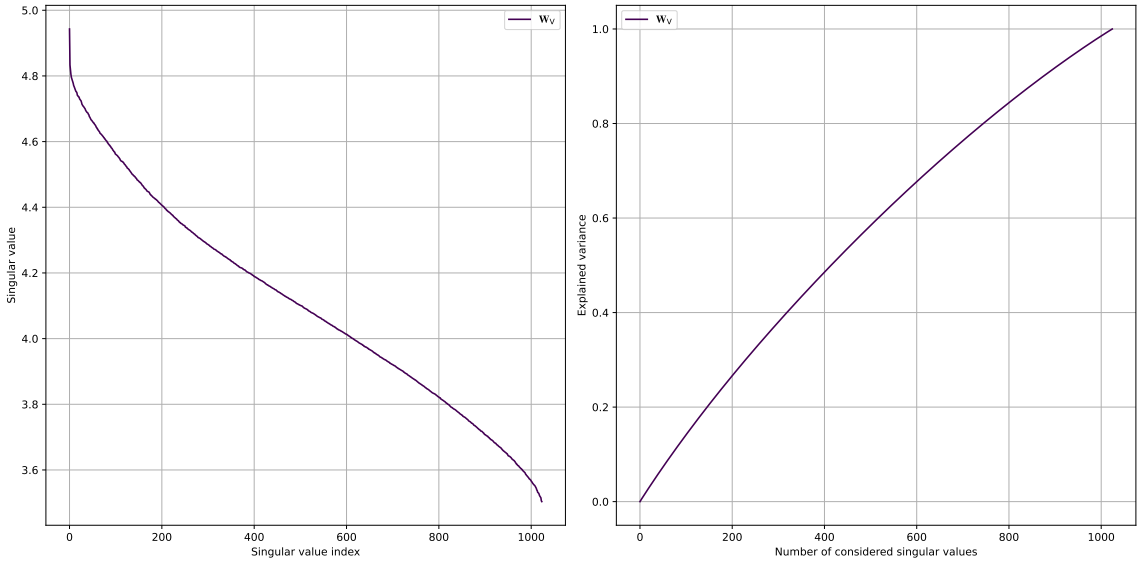


Figure 7.14: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention value matrices in Llama 3.1 blocks.

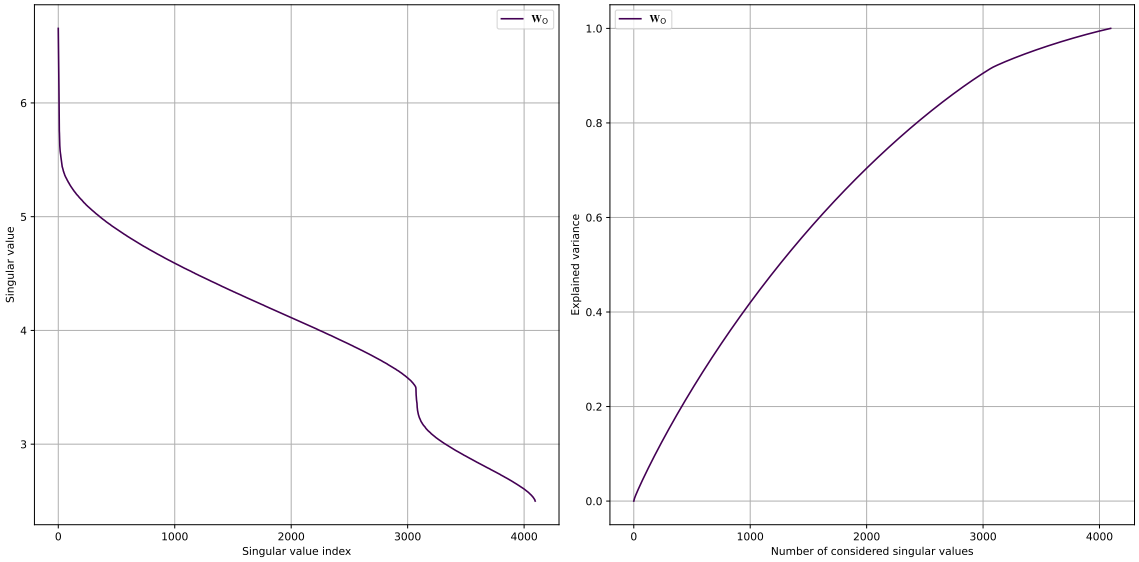


Figure 7.15: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention output matrices in Llama 3.1 blocks.

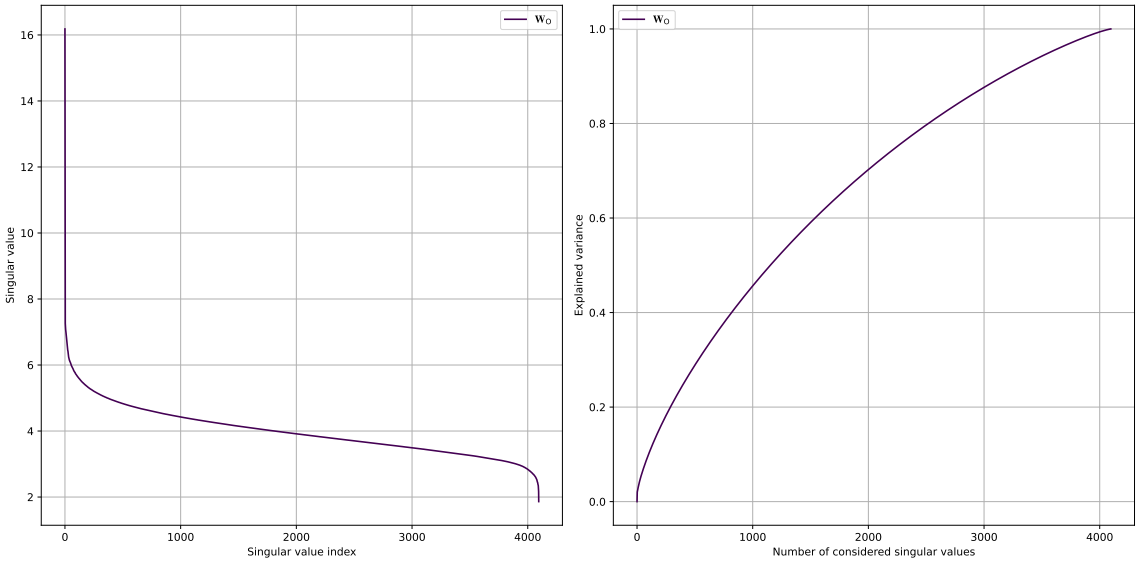


Figure 7.16: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention output matrices in Llama 3.1 blocks.

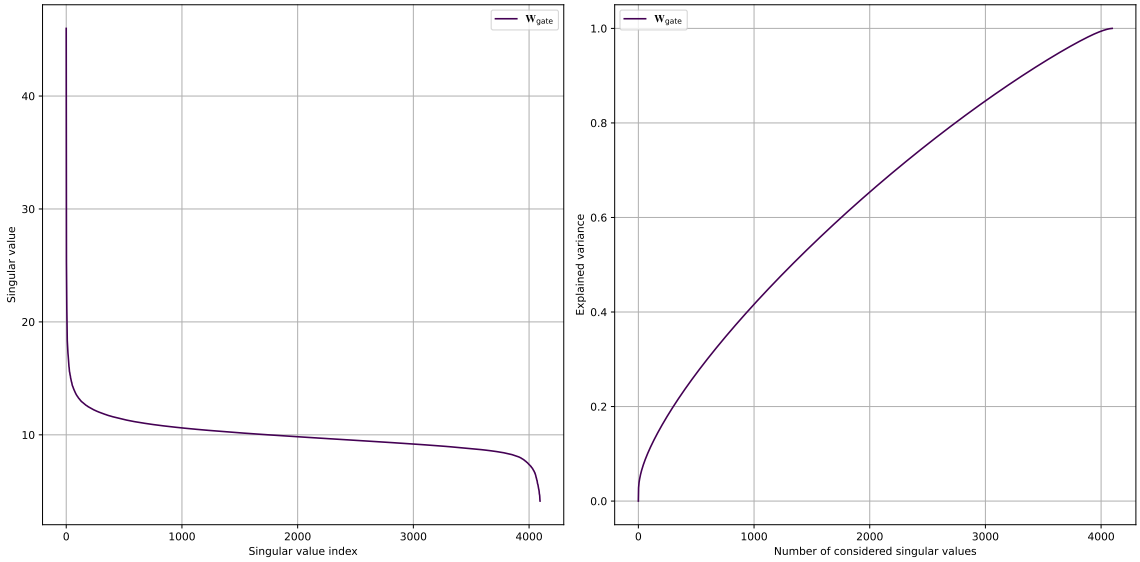


Figure 7.17: Singular value distribution and cumulative explained variance of the row-wise concatenation of gate projection matrices in Llama 3.1 blocks.

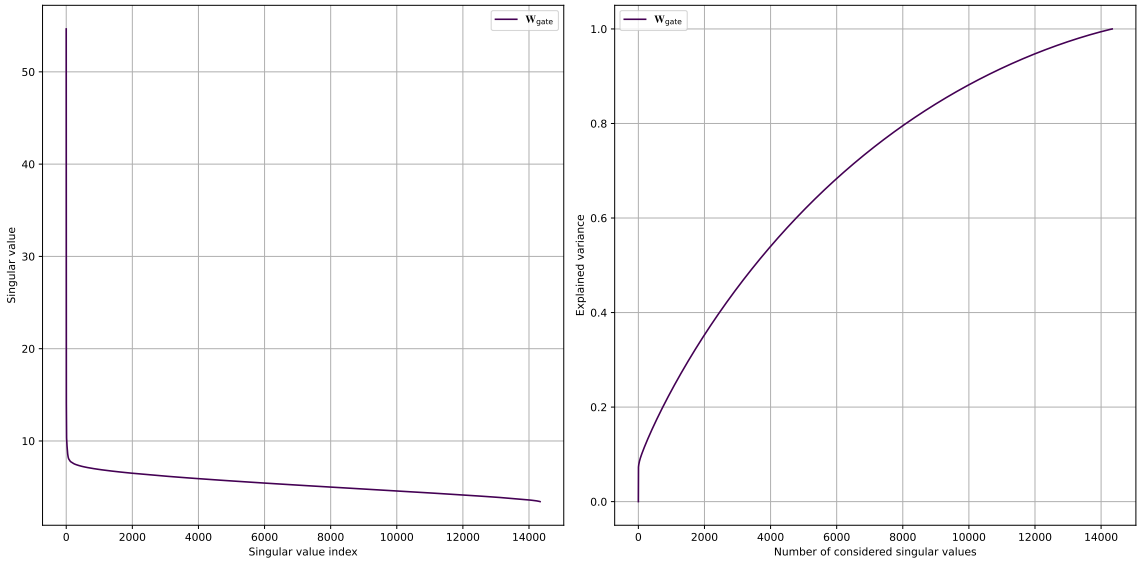


Figure 7.18: Singular value distribution and cumulative explained variance of the column-wise concatenation of gate projection matrices in Llama 3.1 blocks.

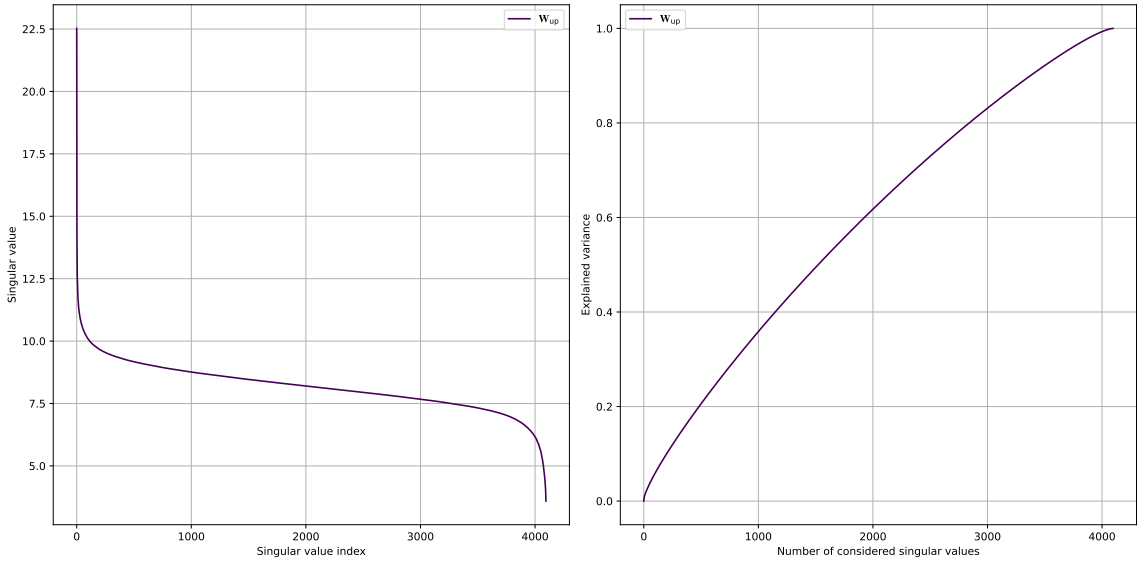


Figure 7.19: Singular value distribution and cumulative explained variance of the row-wise concatenation of up projection matrices in Llama 3.1 blocks.

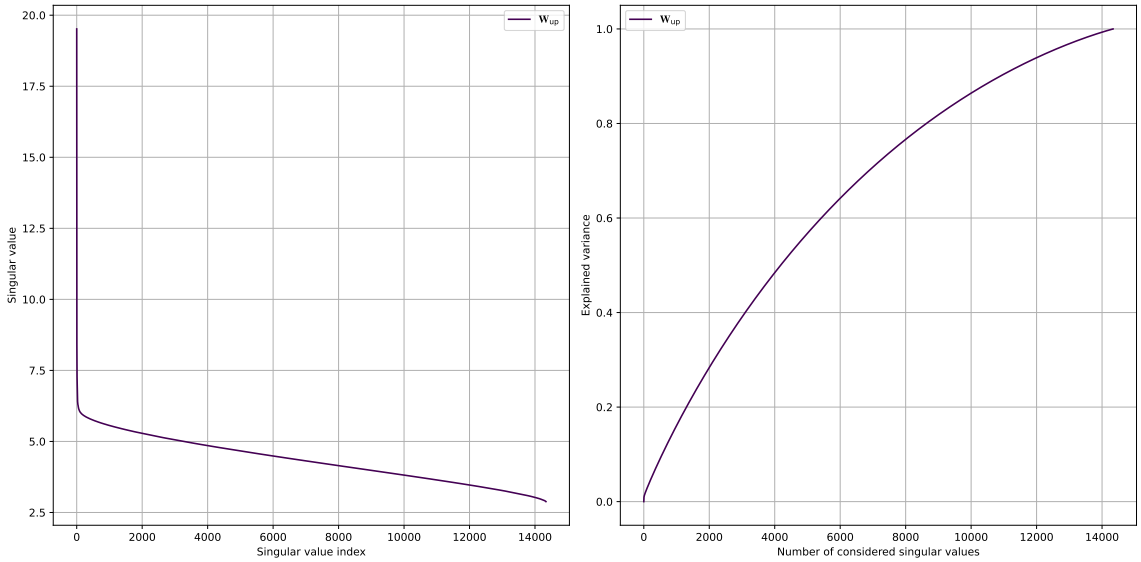


Figure 7.20: Singular value distribution and cumulative explained variance of the column-wise concatenation of up projection matrices in Llama 3.1 blocks.

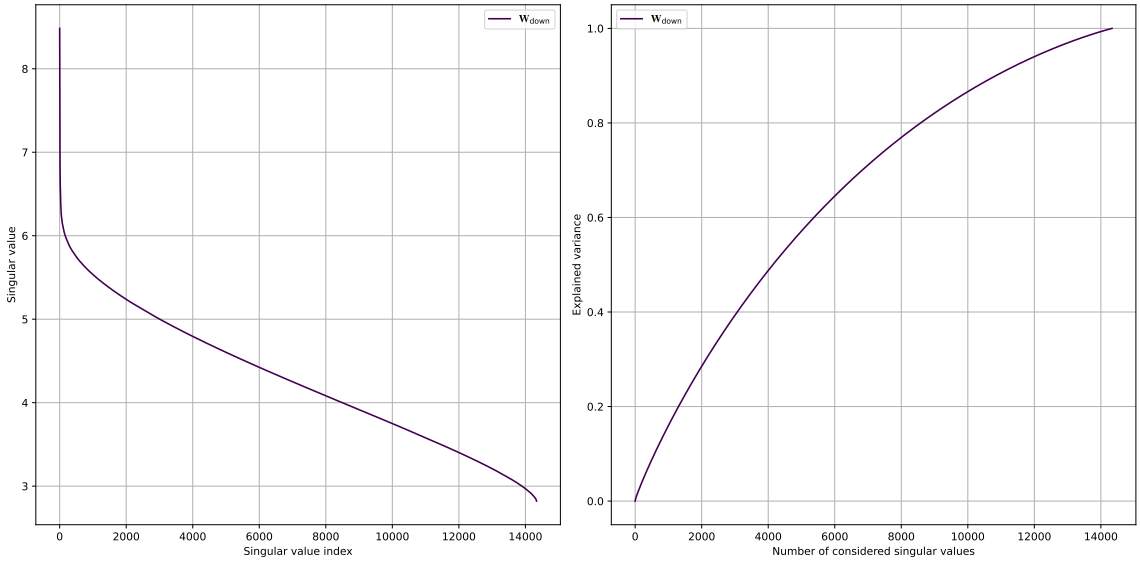


Figure 7.21: Singular value distribution and cumulative explained variance of the row-wise concatenation of down projection matrices in Llama 3.1 blocks.

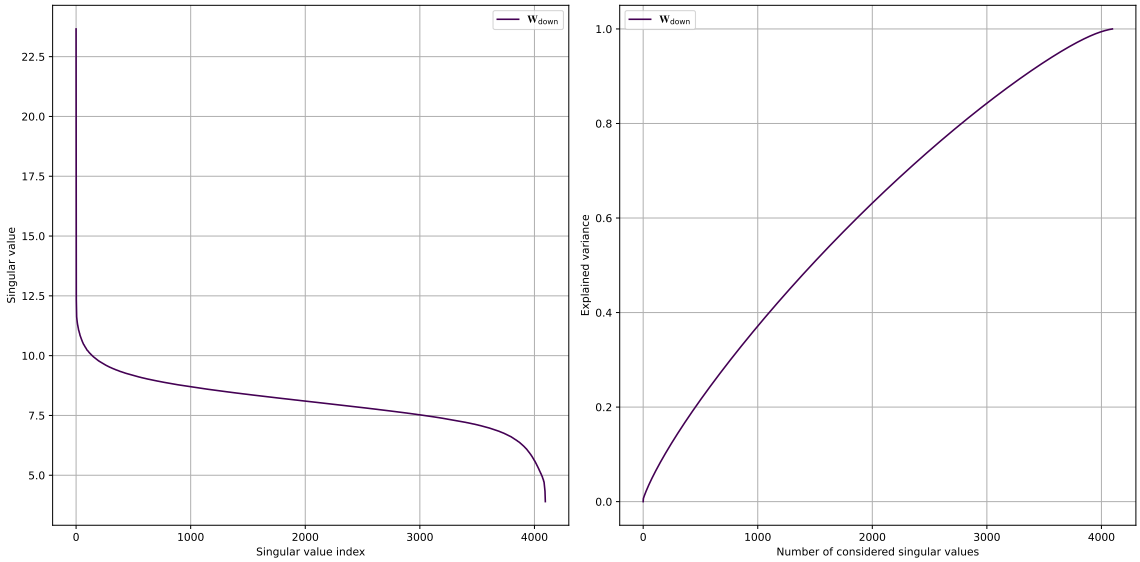


Figure 7.22: Singular value distribution and cumulative explained variance of the column-wise concatenation of down projection matrices in Llama 3.1 blocks.

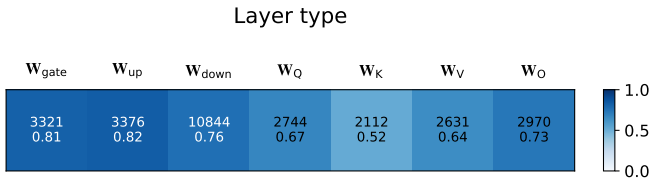


Figure 7.23: Approximate ranks of the weight matrices in Llama 3.1 row-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

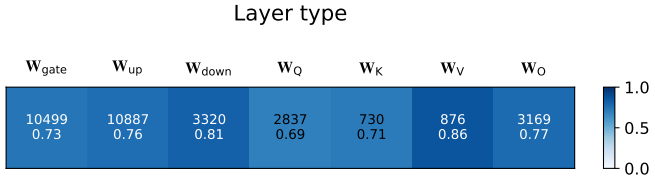


Figure 7.24: Approximate ranks of the weight matrices in Llama 3.1 column-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

7.1.3. Inter-Sub-Block Replacement Analysis

A crucial aspect of redundancy examined in this thesis is functional redundancy. Functional redundancy refers to cases where multiple layers, sub-blocks, or blocks perform highly similar or even nearly identical transformations on the input.

Given two sections of the Transformer, where the first computes a function f and the second computes a function f', the objective is to assess whether $f \approx f'$ and quantify their degree of similarity or redundancy.

How to Compute Functional Similarity

The most rigorous approach to evaluating the functional similarity between two transformations would be to compare their closed-form expressions, analyzing the similarity of their outputs when applied to identical inputs.

Although this method is feasible for simple transformations within small domains, it becomes impractical for the complex, high-dimensional transformations performed internally by the Transformer. Specifically, self-attention mechanisms and feed forward networks

involve non-linear composite functions operating over vast input spaces, making direct analytical comparison intractable. Additionally, the high-dimensionality of Transformer embeddings makes exhaustive output comparisons computationally prohibitive. Lastly, random inputs offer limited insight as they do not originate from meaningful sentences that more accurately reflect the real-world behavior of the model.

A notable exception arises in the case of single linear transformations. Comparing one linear layer with another is both straightforward and effective, as the difference between their weight matrices suffices to quantify similarity. This eliminates the need to compute the transformations on actual inputs.

The primary challenge in comparing transformations across different modules lies in defining a valid metric of similarity.

Within this analysis, various methods for evaluating similarity between modules can be considered. These methods differ in their suitability for layer-level and higher-level comparisons. Several strategies emerge as potential candidates:

- **Norm of the difference between the weights in the two modules**: This method quantifies divergence by computing the norm of the difference between the parameters of two modules. It is computationally efficient and provides a clear measure of dissimilarity between individual weight matrices. However, it is restricted to direct parameter comparison and does not account for composition of functions or non-linear transformations when applied to sequences of layers, which may lead to a misleading assessment of functional similarity.
- **Cosine similarity of the activations, given a calibration set**: Cosine similarity is computed between the output activations of two modules for the same inputs from a calibration set, providing a clear and informative measure of similarity. If the average cosine similarity is close to one, the functions perform similar transformations; conversely, if it deviates significantly from one, they are dissimilar. In domains such as those of Transformer layers, similarity tends to be inherently low due to the high dimensionality of word vectors. The statistical significance of this approach depends on the dataset size relative to that of the embedding space, and it does not account for activation scale, which may be relevant in certain contexts.
- **Norm of the difference between the activations, given a calibration set**: This method measures the dissimilarity between two modules by computing the norm of the difference between their output activations for the same inputs from a calibration dataset. If the norm is close to zero, the modules exhibit highly similar

behavior. Although this approach effectively quantifies differences, it shares the limitations of cosine similarity, including dependence on the dataset size and challenges associated with high-dimensional embeddings, which may impact statistical significance.

- **Global performance measure when a module is replaced by another**: This approach assesses the similarity between two modules within a Transformer architecture by replacing one with the other and evaluating the impact on the overall performance of the model. Functional similarity is inferred by comparing the performance of the modified model with the original. The method can be trivially extended to scenarios where multiple layers are replaced simultaneously following a predefined mapping strategy. A key advantage of this method is that performance benchmarks used for evaluation are robust and provide informative assessments of the quality of large language models. Furthermore, they yield single-metric evaluations that are straightforward to interpret. Although this provides a practical assessment under real operational conditions, it remains susceptible to noise, as the observed variations reflect changes across the entire model rather than isolating the specific effect of the exchanged modules. Additionally, although less than other metrics, the reliability of this approach is still constrained by its dependence on the benchmarking dataset.

After some experimentation with all the metrics, our experiments centered on the last measurement technique, which involved swapping modules from one part of the model with those from other parts. By evaluating the performance of the resulting model, this analysis aims to determine the extent of functional overlap and redundancy in module computations. This method specifically helps us investigate whether different components within the model perform similar transformations on the input or contribute unique and indispensable computations.

Methodology to Evaluate Functional Redundancy

The analysis follows these steps:

1. **Layers Replacement**: A selected group of layers or higher-level modules is replaced with another group following a predefined mapping strategy. For instance, self-attention layers in block 1 of the Transformer stack can be replaced with their counterparts from block 2.

   Various replacement strategies can be employed; however, to address the research questions guiding this thesis, the selected approach for evaluating similarity across

the B blocks of a model involves single block-pair replacement. Specifically, layers or modules in block i are replaced with their counterparts from another block $i+k$, where $k \in [-i, -i+1, \ldots, B-i]$. This ensures that the layers of each block are individually replaced with those of every other block, allowing for a systematic evaluation of similarity between pairs of elements.

2. **Model Evaluation**: The modified model is assessed using benchmarks to measure the impact of module replacement on overall performance.

   If the performance of the model remains stable despite the interchange of layers, this suggests that exchanged modules are executing redundant computations. Such findings could offer valuable insights into potential model optimization strategies, such as pruning or merging similar layers to reduce the parameter count without sacrificing accuracy or performance.

   These evaluations leverage well-established benchmarks, discussed in Section 5.3, selecting tests designed to capture the impact of layer modifications on the core language processing capabilities of the models. Since each benchmark presents distinct challenges and characteristics, performance variations across different tasks are analyzed to ensure robustness in the results. This comparative approach also provides a deeper understanding of how redundancy levels fluctuate depending on task complexity and the nature of the computations performed by the model.

Although other modules were considered, the analysis primarily focused on replacing the feed forward neural network components within Transformer blocks. The hypothesis was that MLPs sections exhibit high redundancy due to their large number of parameters and the nature of their computations. Moreover, the intrinsic structure of the FFNN module, where weight matrices can be internally reordered without altering the output function, provides an additional motivation for evaluating the functional similarity of these modules across blocks.

Given an input embedding $\mathbf{x}$, the function at block i of a three-layer feed forward forward neural network module is expressed as:

$$\mathbf{FFNN}_i = \left(f_a\left(\mathbf{x} \cdot \mathbf{W}^T_{\text{gate},\,i}\right) \circ \left(\mathbf{x} \cdot \mathbf{W}^T_{\text{up},\,i}\right)\right) \cdot \mathbf{W}^T_{\text{down},\,i} \tag{7.2}$$

The objective is to determine whether the FFNN functions in different layers exhibit approximate equivalence:

$$\mathbf{FFNN}_i \approx \mathbf{FFNN}_j \tag{7.3}$$

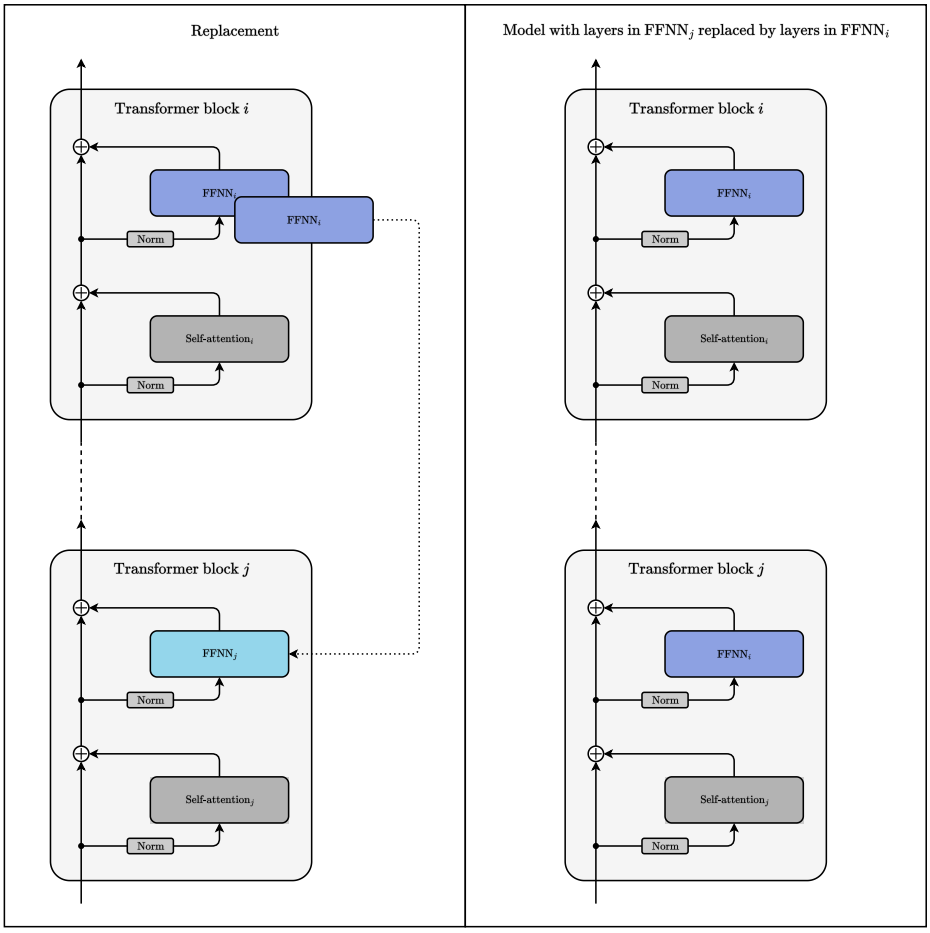


Figure 7.25: Diagram illustrating the replacement of feed forward neural network modules during the analysis. The layers in the FFNN of Transformer block j (light blue) are replaced with those from the FFNN of block i (blue).

Experimental Setup

The framework used for evaluation is the **Language Model Evaluation Harness** library [20], a tool developed by EleutherAI to standardize and simplify the assessment of language models across a variety of natural language processing tasks. The library integrates seamlessly with popular model repositories such as Hugging Face.

The first benchmark employed in the redundancy evaluation is HellaSwag (Section 5.3.2). This benchmark tests the comprehension and basic reasoning capabilities of the model.

The main analysis is conducted on Llama 3.1 with 8 billion parameters (Section 5.4.1) and Gemma 2 with 2 billion parameters (Section 5.4.3). In this section, plots for the Llama model are presented, while the corresponding diagrams for the Gemma model are provided in Appendix B.

Characterization of the Similarity

The results are visualized as heatmaps, with columns representing the layers being replaced and rows denoting the layers used as replacements, both explicitly labeled.

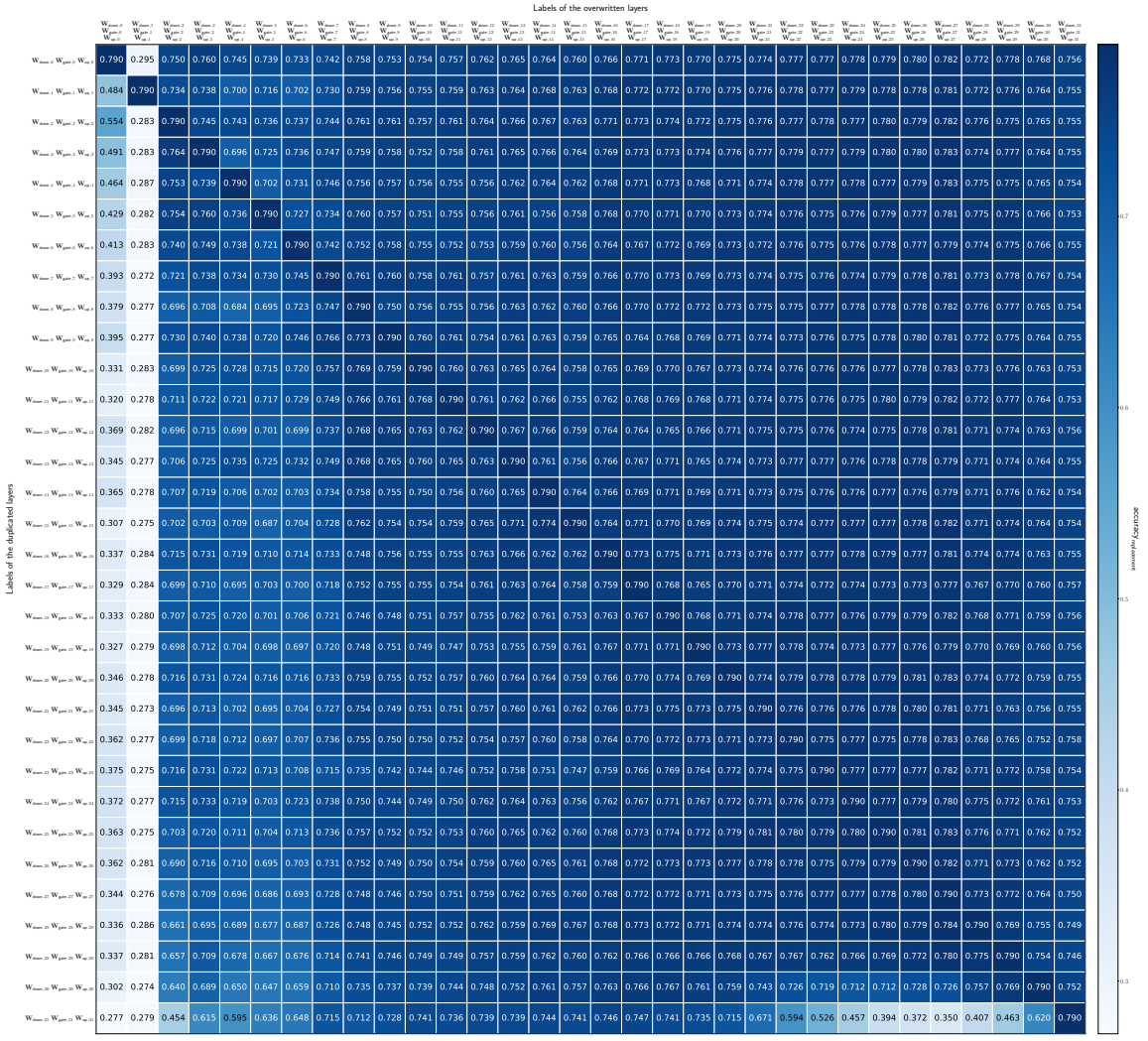


Figure 7.26: Heatmap illustrating the accuracy on HellaSwag of Llama 3.1 having linear layers of the FFNN module replaced by the ones of a different block.

For example, in Figure 7.26, the cell at the intersection of the row labeled $\mathbf{W}_{\text{down},0}$ $\mathbf{W}_{\text{gate},0}$ $\mathbf{W}_{\text{up},0}$ and the column labeled $\mathbf{W}_{\text{down},1}$ $\mathbf{W}_{\text{gate},1}$ $\mathbf{W}_{\text{up},1}$ represents the performance of a model in which the down, gate, and up projection matrices of block 1 in the FFNN module are replaced with their counterparts from block 0. This operation effectively replaces the entire FFNN module of block 1 with that of block 0, as all its parameters are

substituted.

A notable detail is that the values in the cells of the main diagonal correspond to the performance of the original model, as the layers are replaced with themselves.

In the case of Llama 3.1, replacing the first few FFNN modules leads to significant performance degradation, rendering the model unable to handle effectively the task at hand. As expected, early transformations play a crucial role in the functionality of the model and exhibit high sensitivity to modifications. The plot suggests that initial layers differ significantly from the others, likely performing distinct operations essential to early-stage processing. In contrast, overwriting FFNN modules in later blocks has a significantly smaller impact. As seen in the plots, these substitutions result in only minor performance variations, with layers closer to the first two being slightly more affected than the later ones. This pattern could suggest a degree of redundancy or similarity among these layers.

A noteworthy observation arises when the last FFNN is used as replacement. Its substitution in place of another module results in a sharp and unexpected performance decline across a wide range of overwritten blocks. This behavior suggests that the final feed forward neural network is highly specialized, performing distinct functions that are not easily transferable to other FFNNs.

A more critical analysis of the plots highlights that the results on HellaSwag do not necessarily confirm functional redundancy among modules. Instead, the findings also suggest the possibility of contextual uselessness in the middle and final blocks. Layers in these blocks may not introduce substantial modifications to the embeddings passing through the Transformer stack, remaining unused for the tasks at hand. Also in this case, as in presence of redundancy, any replacement would not result in a significant drop in performance.

Starting from this observation, effectively assessing functional redundancy requires two key elements:

1. **More comprehensive and challenging benchmarks**: The benchmarks must be sufficiently demanding to ensure that all layers, including the middle ones, actively contribute to meaningful computations. Simpler tasks may fail to engage these layers, leading to an inaccurate assessment of their functional similarity.

2. **A reliable baseline for performance comparison**: Establishing a baseline is crucial to evaluate whether each modification introduced in the analysis has a meaningful effect on performance relative to some minimum performance. The goal is to determine whether high accuracy values in specific replacement cases arises from

similarity between modules or simply from the limited involvement of the replaced modules in the given task.

To address the need for more challenging benchmarks, TruthfulQA (Section 5.3.1) and GSM8K (Section 5.3.3) are introduced. GSM8K assesses performance on complex grade-school mathematical word problems, while TruthfulQA challenges the model by evaluating its ability to generate factually accurate responses, specifically testing its resistance to common misconceptions and adversarially designed questions.

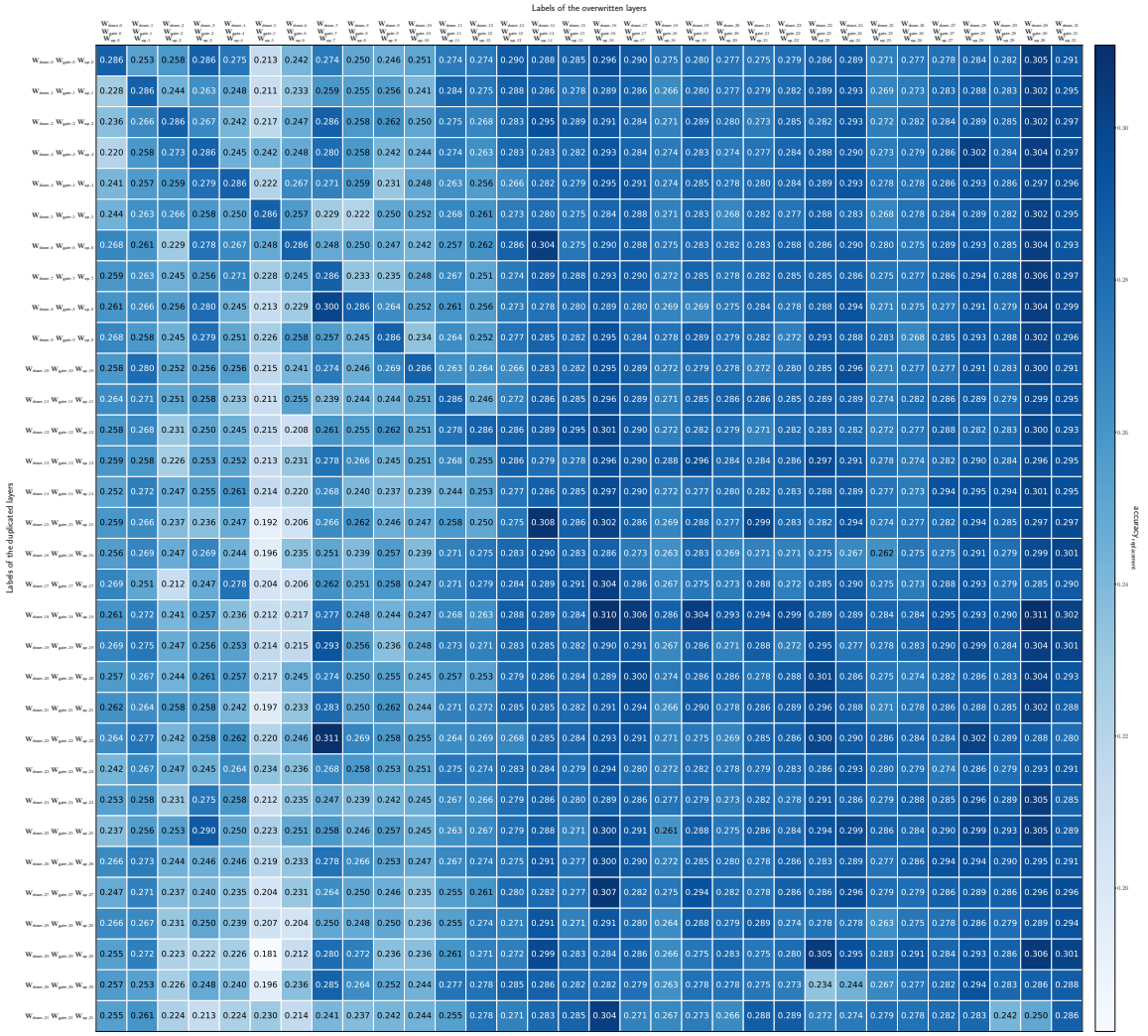


Figure 7.27: Heatmap illustrating the accuracy on TruthfulQA of Llama 3.1 having linear layers of the FFNN module replaced by the ones of a different block.

The results on TruthfulQA are challenging to interpret, as the model performs poorly

on this benchmark. Performance is close to that of a random classifier, and oscillations in accuracy around 0.25 may be considered noise rather than meaningful information. The initial blocks, up to approximately the thirteenth, appear to be the most affected by replacement, while substituting the later ones does not significantly impact the quality of the model. Although the initial layers were also the most critical in HellaSwag, the pattern of performance degradation differs between the two benchmarks. Thus, despite the limited robustness of the TruthfulQA evaluation, this discrepancy suggests that replacement configurations preserving performance in HellaSwag do not necessarily imply functional similarity.

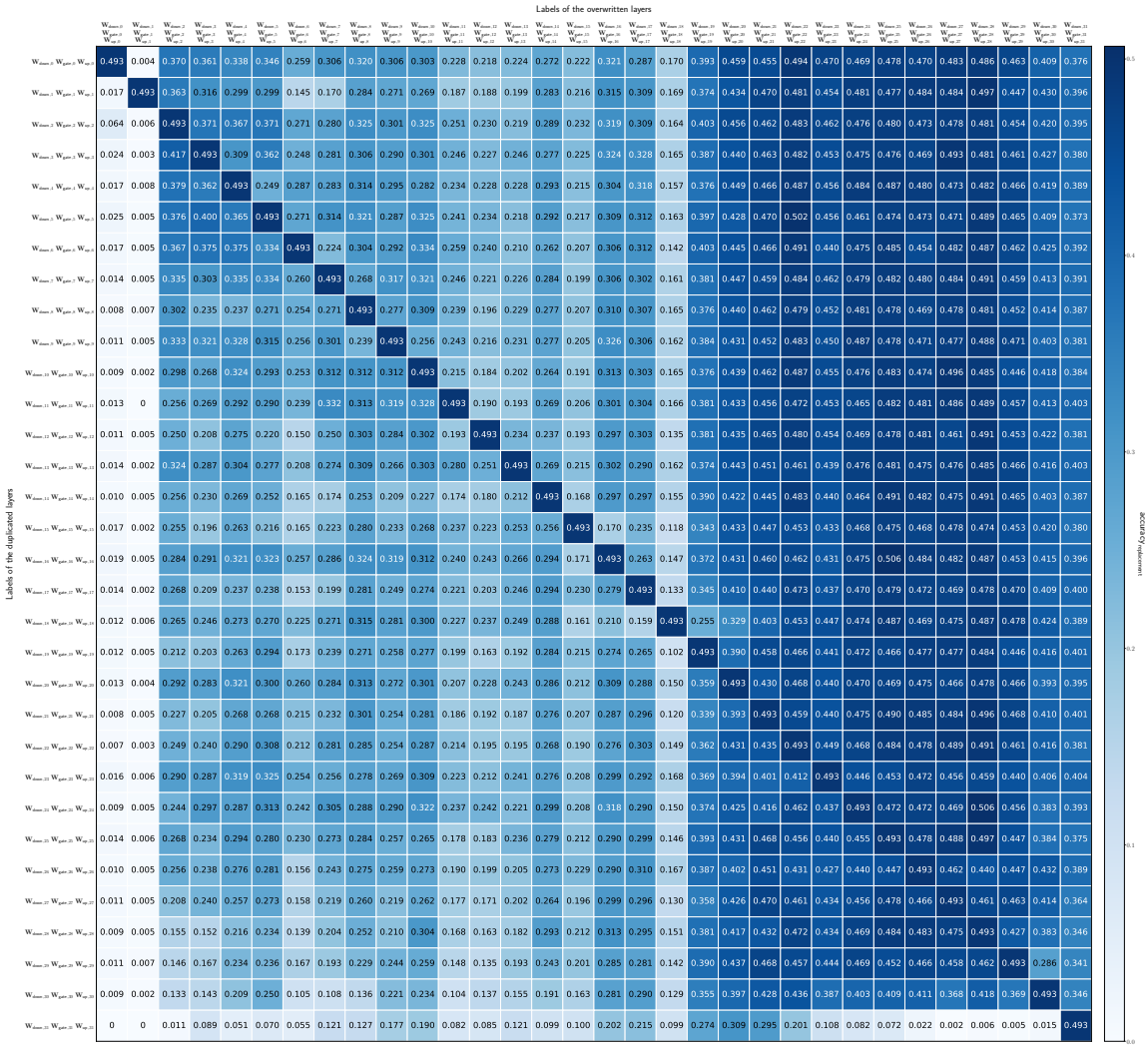


Figure 7.28: Heatmap illustrating the accuracy on GSM8K of Llama 3.1 having linear layers of the FFNN module replaced by the ones of a different block.

The heatmap of the GSM8K results reveals a extreme degradation in performance when one of the first two blocks is modified, similar to what is observed on HellaSwag. These findings confirm their critical importance and further suggest a potentially high divergence between the functions computed by the FFNN module in the first blocks compared to those in later ones. The plot also shows a significant drop when layers from blocks in the first half of the Transformer stack are substituted. Conversely, replacing one of the final Transformer blocks has a negligible impact on performance. Unlike simpler benchmarks, GSM8K imposes a higher cognitive load on the model, emphasizing the crucial role of middle layers in computations within certain contexts.

These results, particularly the discrepancies with those obtained on HellaSwag, suggest that functional differences also exist among the middle blocks and reinforce the hypothesis that performance retention in HellaSwag was likely not due to redundancy but rather to contextual uselessness. This distinction becomes evident in this case, as these layers are now engaged in meaningful computations.

The differences between Figure 7.26 and Figure 7.28, along with the weaker observations from Figure 7.27, prevent the formulation of definitive conclusions regarding layer similarity. This lack of coherence underscores the need for further investigation. A comparison with the baseline remains essential for accurately assessing actual redundancy.

Ablation Study as Baseline

To establish a baseline for evaluating module importance and redundancy, an ablation study is conducted. Instead of replacing components, they are entirely removed. By analyzing the impact of their removal, the goal is to determine whether these layers contribute to meaningful computations or if they are unnecessary in the given context.

The ablation study follows these steps:

1. **Layers removal**: Layer Removal: A selected group of layers is eliminated from the model, ensuring they no longer contribute to any computation. This is achieved by replacing the corresponding layers or modules with a function that outputs zero vectors. Specifically, in the experiments, this is implemented by setting all weights of the targeted module to the null matrix W_{zeros}. If a bias term were present, it would also be set to zero, though this is not the case in the analyzed models. For instance, removing the FFNN of block i is accomplished by setting $\mathbf{W}\text{down}, i$, $\mathbf{W}\text{gate}, i$, and $\mathbf{W}_{\text{up},i}$ to zero. This ensures that the residual embedding passing through the stack remains unaltered by the module. This approach aligns with the substitution experiments, enabling a direct comparison between the two methods.

2. **Model Evaluation**: The modified model is evaluated using the same benchmarks as in the previous analysis to quantify the impact of layer removal on performance.

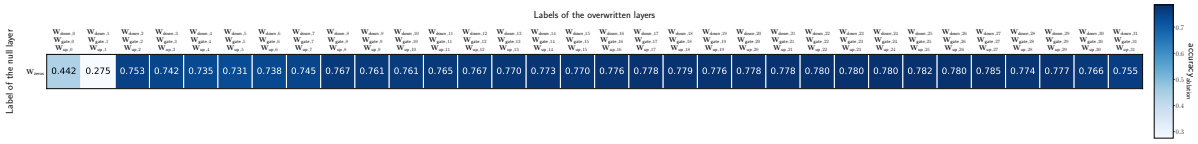


Figure 7.29: Heatmap illustrating the accuracy on HellaSwag of Llama 3.1 having linear layers of the FFNN module removed.

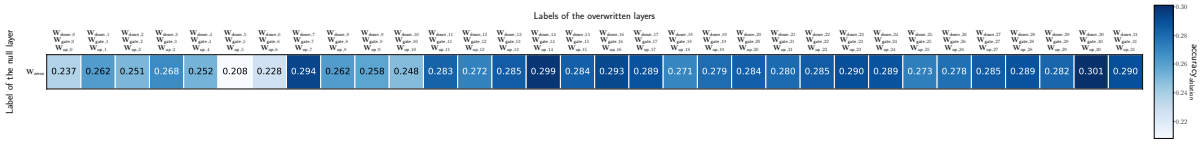


Figure 7.30: Heatmap illustrating the accuracy on TruthfulQA of Llama 3.1 having linear layers of the FFNN module removed.

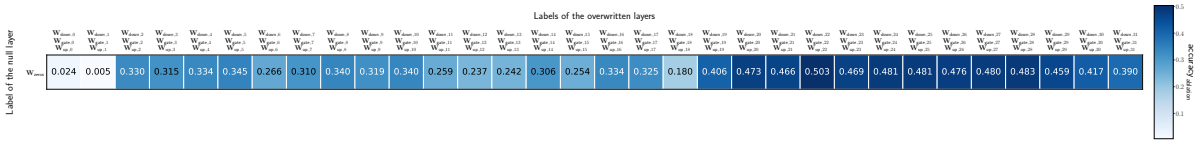


Figure 7.31: Heatmap illustrating the accuracy on GSM8K of Llama 3.1 having linear layers of the FFNN module removed.

Analyzing the ablation plots reveals that they highlight similar considerations to those observed in the layer replacement case, as the performance distribution follows a comparable pattern when FFNN is substituted rather than removed. In particular, the removal of early FFNNs results in severe performance degradation across benchmarks, indicating that these modules are highly engaged in computations and play a crucial role in the functionality of the model. Conversely, modifications to later blocks have limited impact, reinforcing the conclusion that these layers are less involved in the task.

Comparing Replacement and Ablation

To assess whether replacement is advantageous, comparable, or inferior to ablation, the performance difference between the two scenarios is evaluated. This difference is computed

by subtracting the performance observed when layers in block i are ablated from the performance obtained when layers in block i are replaced by those in block j. Graphically, this corresponds to subtract from each cell of the heatmap representing the performance of the replacement case the value in the corresponding column of the ablation study.

A significant positive deviation in this difference suggests the presence of meaningful redundancy, indicating that the introduced layer can approximate the functionality of the substituted one more effectively than a null layer. Conversely, if the performance of the replaced model closely aligns with that of the ablated model or if replacement leads to a greater degradation, it implies that the replaced layer likely performs unique and essential computations that its counterpart cannot replicate.

The results are presented in a heatmap, where:

- Red cells indicate cases where the replacement of FFNN was less disruptive than its ablation, meaning that substituting a module with another was more favorable than removing it entirely.
- Blue cells indicate cases where ablation resulted in better performance than replacement, suggesting that removing a module was less detrimental than replacing it with a different one.

Examining the diagonal reveals which module removals (ablation cases) cause the greatest performance drop relative to the original model, identifying the most critical components for the task. Higher values indicate greater degradation, while values near zero suggest minimal impact.

The primary focus should be on columns where the diagonal contains high positive values, corresponding to strongly red cells. If a replacement effectively preserves performance, the cells in that column should exhibit similar or at least high values, indicating functional redundancy. Conversely, blue cells in these columns suggest that the replacement causes greater degradation than ablation, highlighting the functional difference between the two modules.

For columns where the diagonal values are close to zero (indicating minimal performance loss in case of module ablation), the presence of red cells is expected to be limited, as replacement is unlikely to produce models significantly better than the original. However, highly negative values in these columns suggest a strong mismatch between the function performed by the original FFNN and that of the replacing one.

A first observation is that the results on TruthfulQA remain noisy and unreliable, with performance often close to random guessing in both the layer replacement and ablation

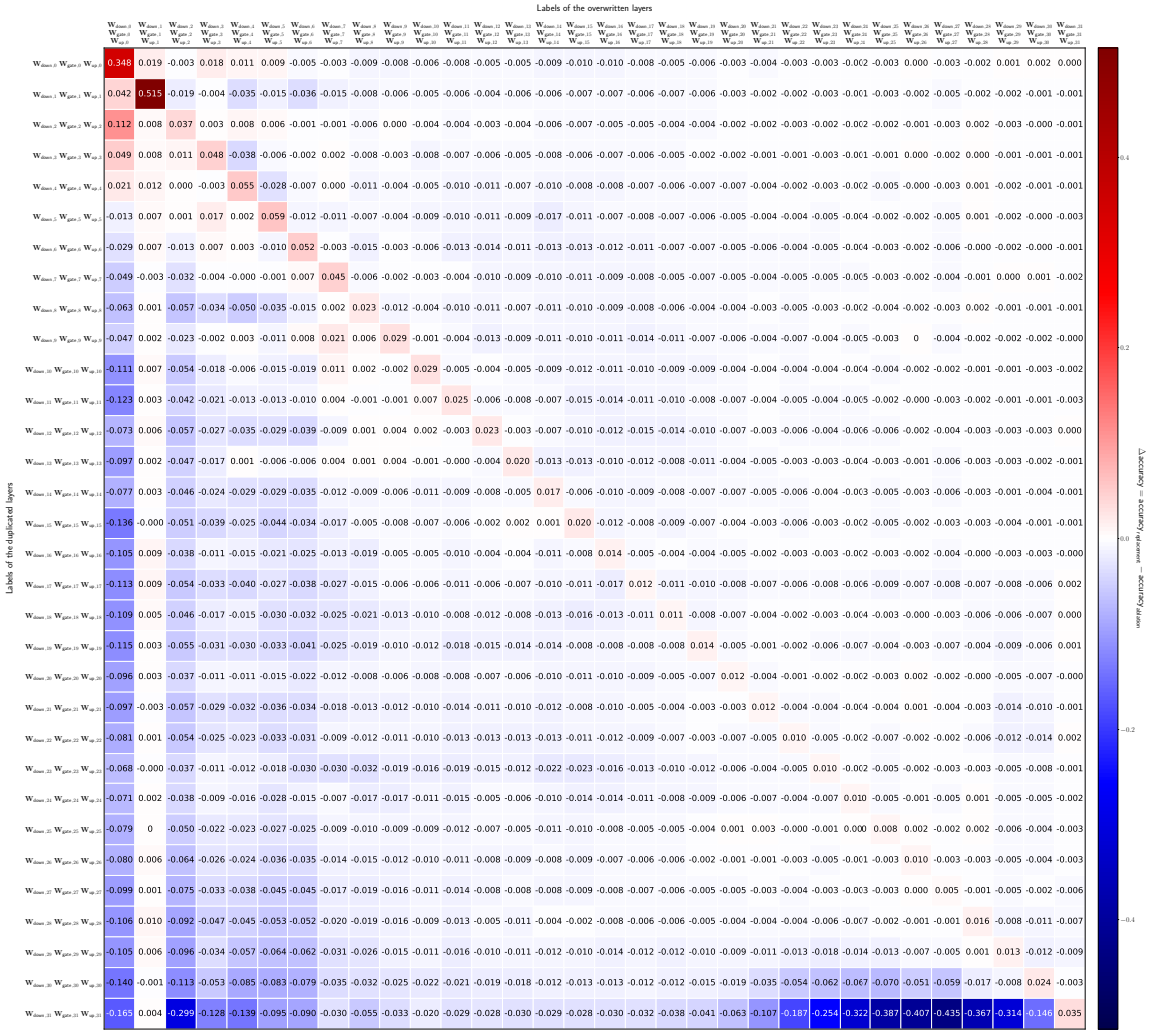


Figure 7.32: Heatmap illustrating the difference in performance evaluated on HellaSwag between Llama 3.1 with linear layers in the FFNN module replaced by those from another block and Llama 3.1 with the same layers removed.

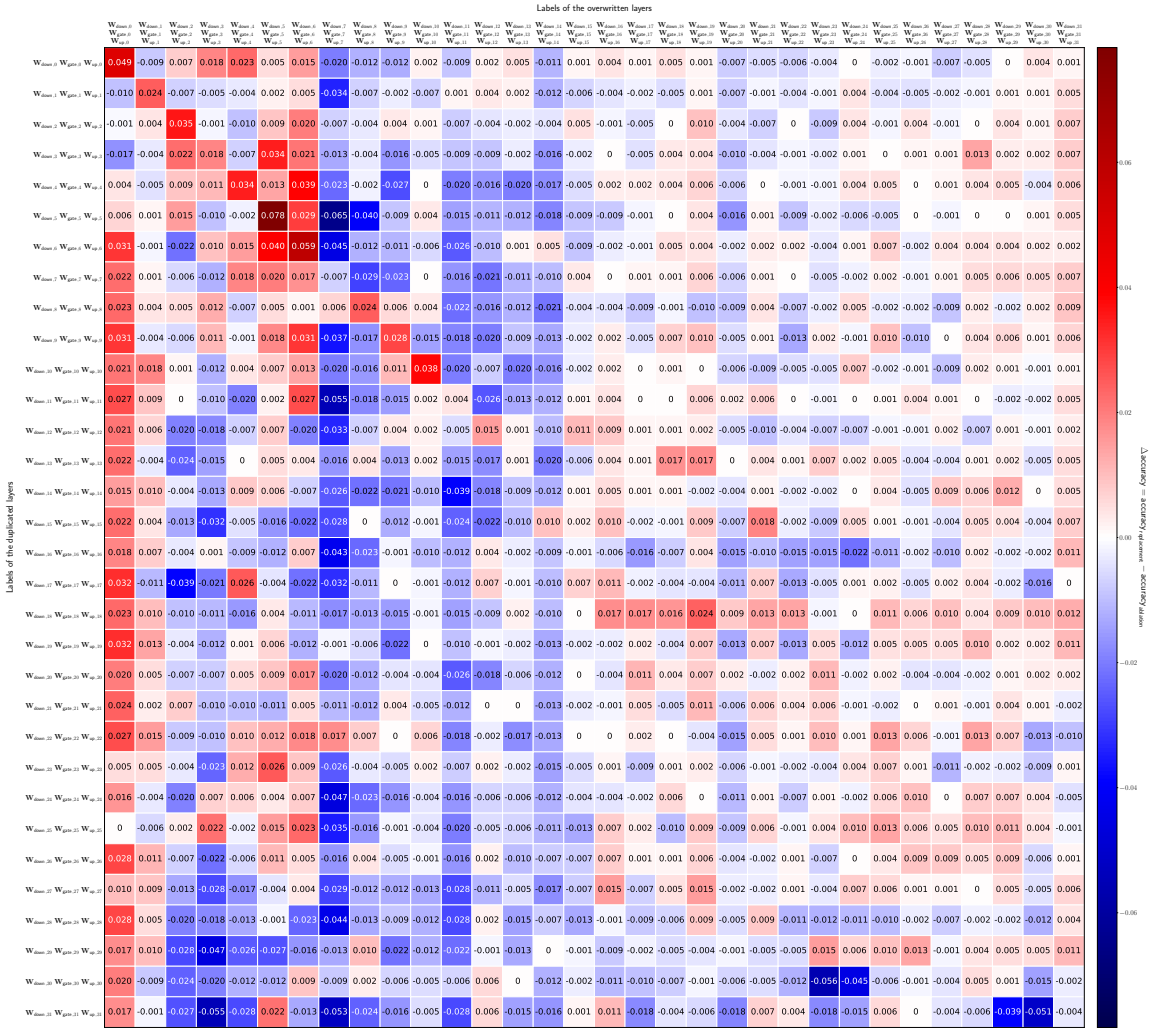


Figure 7.33: Heatmap illustrating the difference in performance evaluated on TruthfulQA between Llama 3.1 with linear layers in the FFNN module replaced by those from another block and Llama 3.1 with the same layers removed.

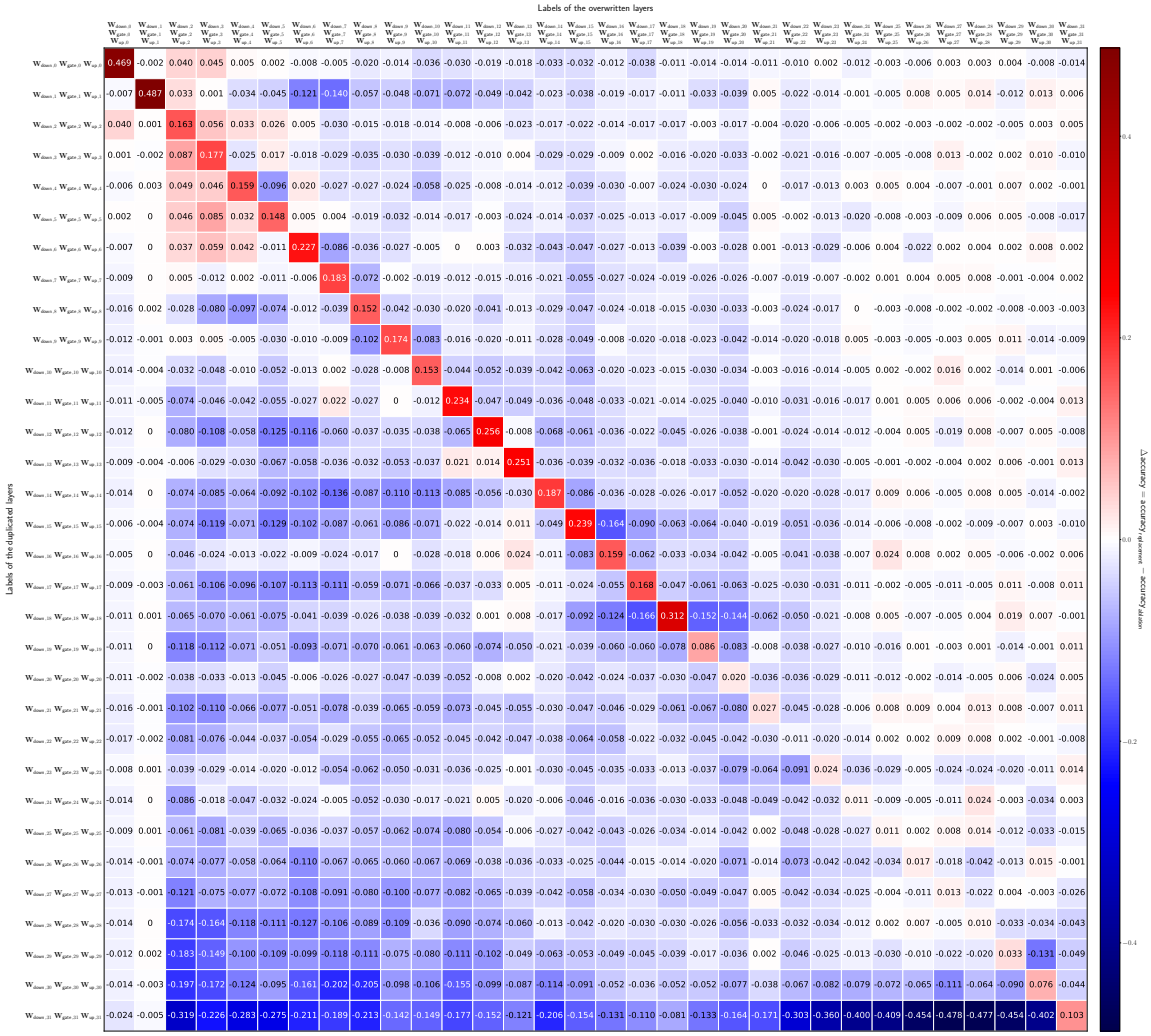


Figure 7.34: Heatmap illustrating the difference in performance evaluated on GSM8K between Llama 3.1 with linear layers in the FFNN module replaced by those from another block and Llama 3.1 with the same layers removed.

cases. The differences between the two scenarios are minimal, and these slight variations may simply result from the limited number of questions the model has answered.

Focusing on the other two plots, their trends appear similar. In many cases, the model with ablated layers performs comparably to the one with substituted layers, with ablation often leading to better performance than replacement, as indicated by the high prevalence of blue cells. Therefore entirely removing the FFNN module is often less disruptive than replacing it with the transformation computed by another block. In other words, preserving the residual connection unchanged in the subsequent layers tends to be more beneficial. Overall, the plots suggest that functional redundancy among FFNN components may be limited.

In columns where the performance delta on the main diagonal is high, some red cells appear near the diagonal, particularly in the early columns. This pattern shows a degree of consistency across the HellaSwag and GSM8K plots, indicating that certain replacements yield high delta values on both benchmarks. Consequently, these pairs of modules can be exchanged with relatively good results and may exhibit partial functional similarity. However, this behavior remains underrepresented, as many other cells in these columns show strongly negative values, reinforcing the idea that most FFNN are distinct.

The transformation performed by the last block stands out as significantly different from those of earlier layers, as replacing other layers with it results in performance that is considerably lower than ablation. This underscores the uniqueness of the FFNN function computed by the final block in Llama 3.1.

Within the limited context of this study, the results indicate that FFNN modules behave differently across blocks and are not interchangeable, with very few exceptions. Therefore, in addressing Research Question 4 (Section 4.4), the analyses indicate that layers in FFNN modules of different Transformer blocks do not exhibit a substantial degree of functional similarity. Specifically, the transformation performed by a module in one block cannot be seamlessly replaced by that of another, as they compute distinct functions. Regarding Research Question 5 (Section 4.5), the heatmaps show some red cells near the diagonal, indicating cases where layers are replaced by those from nearby blocks. However, no consistent pattern emerges, making it difficult to conclude that adjacent FFNNs are inherently more similar than those from further apart blocks.

The heatmaps illustrating performance differences between replacement and ablation results exhibit a pronounced asymmetry, which arises from several factors:

1. Non-equivalent effects of swapping modules between blocks: the replacement of the

FFNN in block i with that of block j and the replacement of the FFNN in block j with that of block i do not necessarily yield identical results due to differences in the embeddings propagated through the network in the two cases. If the modules are highly similar, the resulting asymmetry may be minor. However, in practice, similarity appears to be low. Considering the two modified models, they compute different transformations in blocks i and in blocks j due to the distinct replacements. These differences lead to divergent outputs and, consequently, unequal performance.

2. Varying module impact across blocks: if the FFNN in block i is not crucial for success on a benchmark, any modification to it will have a similarly minimal effect on the performance, resulting in a near-zero difference between the accuracy in the replacement case and in the ablation case. However, when the same module in block i is used to overwrite one from a block essential for computing the correct output, performance may differ from that of ablation, introducing asymmetry in the plots.

Ultimately, some degree of symmetry would be expected in heatmaps comparing performance differences between the replacement and ablation cases, provided that the paired modules are sufficiently similar, particularly in components that play a key role in the task. However, when module similarity is low, asymmetry becomes a natural and expected outcome.

This study focuses on inter-sub-block redundancy. The results emphasize the complexity of redundancy in large language models, showing that identifying and isolating functionally redundant components remains challenging while suggesting that functional redundancy may be limited across FFNN of different blocks. Nonetheless, this does not preclude the possibility that other recurring features, beyond straightforward functional redundancy, could be leveraged for model compression.

Experiments on other models and modules

This section presents the results of the layer replacement analysis conducted on models other than Llama 3.1 and on modules beyond the FFNN. Detailed plots and additional information are provided in Appendix B.

The analysis of functional redundancy in the self-attention mechanism of Mistral 3.1 suggests a partial degree of similarity across self-attention modules, as replacing them with those from other blocks often results in better performance than their ablation. This trend is particularly evident in GSM8K, where many replacements are less disruptive than removing the layers entirely. The main exception is the first blocks, where attention appears highly specialized and functionally distinct from the rest. However, due to the

limited scope of the experiments, the marginal improvement of module replacement over the weak baseline, and the lack of consistent patterns, these findings should be interpreted with caution. Additionally, whether modules in adjacent blocks exhibit greater similarity remains an open question.

Analogous analyses to the ones conducted on Llama 3.1 are performed on the Gemma 2 (2B) model.

The analysis of FFNN replacement in Gemma 2 (2B) shows that substituting these modules across blocks generally degrades performance more than removing them entirely. A similar trend is observed in the self-attention mechanism, where replacement typically results in performance degradation comparable to or worse than ablation.

Although a few exceptions, primarily in self-attention but also in FFNN, show slight improvements suggesting partial redundancy, these cases are limited. Overall, the findings indicate that the transformations of the modules do not generalize well across blocks, reinforcing the notion of minimal inter-sub-block functional redundancy.

The comparison between Llama 3.1 and Gemma 2 (2B) reveals key differences. Gemma 2 (2B) exhibits lower redundancy, as indicated by stronger negative deltas between replacement and ablation performance. This may be due to its smaller size, which likely results in less over-parameterization and fewer duplicated computations. The differences in knowledge distribution and learning patterns highlight how training processes, architecture, and dataset composition uniquely shape the reasoning mechanisms of each model.

7.2. Experiments on Compression Methods

This section presents the experiments conducted to evaluate the compression techniques introduced in Chapter 6, along with an analysis of their outcomes. Following a description of the experimental setup, relevant tables and figures illustrate the results, which are subsequently discussed in detail.

The effectiveness of the proposed methods is assessed using established benchmarks outlined in Section 5.3, specifically HellaSwag and GSM8K, which provide a rigorous and robust evaluation framework.

All experiments were conducted on a server equipped with 64 GB of RAM and an Nvidia GeForce RTX 4090 GPU with 24 GB of VRAM.

7.2.1. Overview of the Experiments

The conducted experiments aim to evaluate the ability of the following methods to compress a large language model while preserving its performance:

1. MASS;
2. GlobaL Fact;
3. ABACO.

All experiments adhere to a similar protocol. Each method requires fine-tuning, either to recover the original performance, as in the case of MASS and GlobaL Fact, or because retraining is an inherent part of the approach, as with ABACO.

The experimental procedure consists of several steps. First, each method undergoes a preparation phase specific to its implementation. After this initial step, the model is evaluated on a set of standardized benchmarks. Next, fine-tuning is performed using a selected training dataset, following one of the approaches outlined in Section 2.4.3: full fine-tuning, partial fine-tuning or adapter-based fine-tuning. Finally, the refined compressed model is re-evaluated on the same benchmark suite to assess its effectiveness.

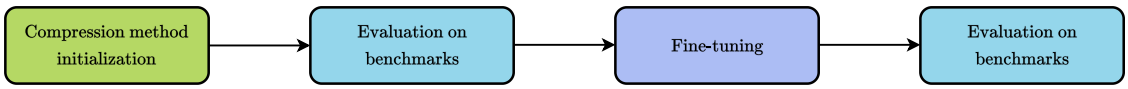


Figure 7.35: Experimental pipeline for applying up and evaluating compression methods.

As in previous experiments, model evaluation is conducted using the automated tool **Language Model Evaluation Harness**.

For the three tested compression approaches, different results are presented, highlighting their respective characteristics and the most relevant performance metrics for each method.

Due to the extensive number of experiments performed and the substantial set of hyperparameters involved in each case, hyperparameter tuning is only partially conducted.

7.2.2. Experiments on MASS

This section presents the experimental setup, results, and key insights derived from evaluating the MASS method.

Experimental Setup

The experiment follows the methodology discussed in Section 7.2.1. The model is firstly compressed using MASS approach and evaluated on HellaSwag (Section 5.3.2) and GSM8K (Section 5.3.3), then fine-tuned on OpenWebText (Section 5.2.2), and subsequently re-assessed on the same benchmarks.

Table 7.1 provides a summary of the key hyperparameters and experimental settings.

Variable	Value
Model	Llama-3.1-8B
Fine-tuning dataset	OpenWebText
Validation dataset	WikiText-2
Batch size	2
Learning rate	10^{-3}
Number of batches	30K
Gradient accumulation steps	2
Optimizer	AdamW
Model precision	16-bit

Table 7.1: Hyperparameters and fine-tuning settings for the experiments conducted on MASS.

The grouping strategy adopted for MASS follows the principles outlined in Section 6.1. Matrices that serve the same function across different Transformer blocks are grouped together. Moreover, the aggregation strategy within each group is the average, as explained in the same section.

Evaluation of Full-Scale MASS

Table 7.2 presents the benchmark performance of the model compressed using MASS on all matrices of the selected type. In this setting, fine-tuning updates only the shared matrix obtained through MASS, while all other parameters remain fixed.

Model	Targets	#Params$_{(CR)}$	HS$_i$	HS$_f$	GSM$_i$	GSM$_f$
Original Model	-	8.030B	0.790		0.493	
MASS	gate	6.210B$_{(77.33\%)}$	0.266	0.259	0	0
MASS	query	7.510B$_{(93.52\%)}$	0.255	0.259	0	0

Table 7.2: Performance of Llama 3.1 (8B) after compression using MASS, where the method is applied to all layers corresponding to a specific role, evaluated on the HellaSwag (HS) and GSM8K (GSM) benchmarks before and after fine-tuning. Results for models before fine-tuning are indicated with the subscript i, while those after fine-tuning are labeled with f.

The application of MASS in its most extreme form, where layers are aggregated and shared across all Transformer blocks based on their functional role, results in severe performance degradation across all benchmarks. The model behaves as a random classifier, indicating a complete loss of language understanding capabilities.

Two key factors may contribute to this failure:

- Overly Aggressive Grouping and Sharing: The aggregation approach may be excessively constraining, as a single shared layer approximates multiple distinct layers across all Transformer blocks. This lack of differentiation may result in an inadequate representation of their diverse functions.
- Suboptimal Aggregation Strategy: Simple averaging fails to capture meaningful structural properties shared by the considered layers. More sophisticated aggregation techniques should be explored to better leverage shared representations. Alternative methods could more effectively preserve essential layer characteristics, provided meaningful similarities exist.

Even after fine-tuning, performance remains at the level of a random classifier, indicating that the model is unable to recover from the degradation introduced by MASS. This suggests that the aggregated representation is too far from a functionally viable parameterization, making adaptation through fine-tuning infeasible. The failure persists even in the case of query compression, which does not substantially alter the parameter count, reinforcing the hypothesis that the compression mechanism itself is fundamentally inadequate.

Table 7.3 reports the training and validation loss recorded during fine-tuning of the model compressed using MASS on all layers of a given type.

Model	Targets	#Params$_{(CR)}$	loss$_{train,1}$	loss$_{train,f}$	loss$_{val,i}$	loss$_{val,f}$	Time
MASS	gate	6.210B$_{(77.33\%)}$	10.38	4.70	13.08	9.31	316 min
MASS	query	7.510B$_{(93.52\%)}$	6.73	6.65	19.46	8.41	291 min

Table 7.3: Training and validation loss of Llama 3.1 (8B) after compression using MASS, where the method is applied to all layers corresponding to a specific role, evaluated during fine-tuning. loss$_{train,1}$ represents the average training loss computed on the samples considered before the first validation epoch, while loss$_{train,f}$ corresponds to the average training loss measured between the last validation epoch and the end of retraining. Instead, loss$_{val,i}$ and loss$_{val,f}$ denote the validation loss before and after fine-tuning, respectively. Additionally, the total fine-tuning time (including validation periods) is reported.

The training loss decreases both when compression targets the gate projection and when targets the query projection but remains relatively high. In the case of query compression, the improvement during training is relatively marginal. Although the validation loss also decreases, as discussed in the following sections, it is not a reliable indicator of actual model capabilities. Indeed, the model fails to achieve meaningful results on the robust benchmarks considered.

Evaluation of MASS Applied to Selected Layers

Building on the insights from the layer similarity analysis (Section 7.1.3), an alternative approach is explored in which only a subset of layers is aggregated, rather than compressing all Transformer blocks. Since the first and last layers were found to be distinct from the rest, they remain unchanged, while only the intermediate layers are subjected to MASS.

Table 7.4 presents the results of the experiment in which MASS is selectively applied to the middle layers, specifically from the 9th to the 24th (indices 8 to 23).

In this refined approach, performance improves compared to the full aggregation scenario, but the model remains significantly degraded. Notably, as expected, performance on HellaSwag is no longer entirely random, as middle layers play a less critical role in this benchmark. However, GSM8K, which depends on deeper layers for reasoning, continues to exhibit zero performance.

Fine-tuning effectively enhances the representation of the shared layer, particularly for HellaSwag, where performance nearly recovers to its original level in the case of query compression and improves significantly when targeting the gate projection. However, for

Model	Targets	#Params$_{(\text{CR})}$	HS$_i$	HS$_f$	GSM$_i$	GSM$_f$
Original Model	-	8.030B	0.790		0.493	
MASS	gate$_{8:23}$	7.149B$_{(89.03\%)}$	0.283	0.503	0	0.001
MASS	query$_{8:23}$	7.779B$_{(96.87\%)}$	0.318	0.701	0	0.010

Table 7.4: Performance of Llama 3.1 (8B) after compression using MASS, where the method is applied to a subset of selected layers corresponding to a specific role, evaluated on the HellaSwag (HS) and GSM8K (GSM) benchmarks before and after fine-tuning. Results for models before fine-tuning are indicated with the subscript i, while those after fine-tuning are labeled with f.

GSM8K, despite a slight improvement after fine-tuning, performance remains substantially below the original. This suggests that deep reasoning abilities, embedded within the middle layers and essential for success in GSM8K, are disrupted by compression and cannot be fully restored through fine-tuning.

Table 7.5 reports the training and validation loss recorded during the fine-tuning of the model compressed using MASS on selected layers.

Model	Targets	#Params$_{(\text{CR})}$	loss$_{\text{train},1}$	loss$_{\text{train},f}$	loss$_{\text{val},i}$	loss$_{\text{val},f}$	Time
MASS	gate$_{8:23}$	7.149B$_{(89.03\%)}$	3.23	2.97	10.96	22.75	277 min
MASS	query$_{8:23}$	7.779B$_{(96.87\%)}$	2.63	2.56	17.10	15.45	255 min

Table 7.5: Training and validation loss of Llama 3.1 (8B) after compression using MASS, where the method is applied to a subset of selected layers corresponding to a specific role, evaluated during fine-tuning. loss$_{\text{train},1}$ represents the average training loss computed on the samples considered before the first validation epoch, while loss$_{\text{train},f}$ corresponds to the average training loss measured between the last validation epoch and the end of retraining. Instead, loss$_{\text{val},i}$ and loss$_{\text{val},f}$ denote the validation loss before and after fine-tuning, respectively. Additionally, the total fine-tuning time (including validation periods) is reported.

The training loss decreases both when compression targets the gate projection and when targets the query projection reaching low values. This behavior indicates successful learning. However, the validation loss remains high and even increases when retraining the gate layer. Although this could be attributed to overfitting, the diversity of the training dataset and the partially strong performance of the fine-tuned models on benchmarks suggest that validation loss may not be a reliable proxy for overall model quality.

Ultimately, these findings partially confirm that the overlap between layers is highly limited in its simplest form, as previously indicated in Section 7.1.3. This reinforces the conclusion that naive parameter sharing strategies struggle to preserve complex hierarchical representations within Transformer architectures.

7.2.3. Experiments on GlobaL Fact

This section presents the experimental setup, results, and key insights derived from evaluating the GlobaL Fact method.

Experimental Setup

The experiment follows the methodology outlined in Section 7.2.1. The model is first factorized and evaluated on HellaSwag (Section 5.3.2) and GSM8K (Section 5.3.3), then fine-tuned on OpenWebText (Section 5.2.2), and subsequently reassessed on the same benchmarks.

Table 7.6 provides a summary of the key hyperparameters and experimental settings.

Variable	Value
Model	Llama-3.1-8B
Fine-tuning dataset	OpenWebText
Validation dataset	WikiText-2
Batch size	2
Learning rate	10^{-3}
Number of batches	32.5K
Gradient accumulation steps	2
Optimizer	AdamW
Model precision	16-bit

Table 7.6: Hyperparameters and fine-tuning settings for the experiments conducted on GlobaL Fact.

The grouping strategy adopted for GlobaL Fact follows the principles outlined in Section 6.2. Matrices that serve the same function across different Transformer blocks, such as all query weight matrices, are grouped together.

Two initialization methods for the global matrices, detailed in Section 6.2.3, are evaluated in the experiments. Specifically, GlobaL Fact with global matrices initialized using

method 1 is denoted as GlobaL Fact$_1$, while initialization with method 2 is labeled as GlobaL Fact$_2$. The exploration of initialization method 3 is reserved for future research.

GlobaL Fact is compared to the model compressed using truncated SVD, with both evaluated and retrained under identical conditions. In both methods, the internal dimension, which controls the level of approximation, must be specified. The factorization of matrices of type t using rank r is denoted as r_t. Since GlobaL Fact achieves greater compression, the rank of the baseline is adjusted to ensure a fair comparison, resulting in a model with a parameter count similar to that of the Global-Local Factorization method.

After exploratory trials with different settings, GlobaL Fact applied to gate and up projections yielded the most promising results; therefore, the experiments focus on these configurations. Nonetheless, further exploration of other elements could provide additional insights.

Evaluation of GlobaL Fact on Benchmarks

Table 7.7 reports the performance of the model compressed using the GlobaL Fact method on benchmark datasets. During fine-tuning, only the factorized matrices are updated, while all other weights remain frozen.

The initial performance of models compressed using GlobaL Fact is significantly lower than that of the original counterpart. None of the tested global matrix initialization strategies provided a stable starting point for the factorization process. Consequently, the compressed models exhibit performance levels on the benchmarks comparable to random guessing, suggesting that the abrupt compression process excessively distorts the underlying representations, rendering the model ineffective for the tasks. However, this degradation is not unique to GlobaL Fact. The truncated SVD approach also results in random-guessing performance levels, highlighting that the challenge lies not solely in the global factorization process but rather in the broader difficulty of initializing factorized models in a way that preserves meaningful representations.

An initial drop in performance of GlobaL Fact relative to non-shared factorization was expected due to the increased approximation error introduced by enforcing a shared global matrix. However, this phenomenon is not observable here, as the SVD-compressed model also suffers from random-guessing performance. A potentially more robust initialization strategy could be achieved through the third global matrix initialization method 3.

When compressing a single layer type, models processed with GlobaL Fact demonstrate partial performance recovery through fine-tuning. This suggests that the factorized model

Model	Ranks	#Params$_{(CR)}$	HS$_i$	HS$_f$	GSM$_i$	GSM$_f$
Original Model	-	8.030B	0.790		0.493	
Truncated SVD	402_{gate}	6.388B$_{(79.55\%)}$	0.262	0.401	0	0.006
GlobaL Fact$_1$	512_{gate}	6.388B$_{(79.55\%)}$	0.257	0.376	0	0
GlobaL Fact$_2$	512_{gate}	6.388B$_{(79.55\%)}$	0.249	0.426	0	0.017
Truncated SVD	402_{up}	6.388B$_{(79.55\%)}$	0.257	0.391	0	0.010
GlobaL Fact$_1$	512_{up}	6.388B$_{(79.55\%)}$	0.267	0.328	0	0.002
GlobaL Fact$_2$	512_{up}	6.388B$_{(79.55\%)}$	0.265	0.337	0	0.002
Truncated SVD	402_{gate} 402_{up}	4.746B$_{(59.11\%)}$	0.265	0.251	0	0
GlobaL Fact$_1$	512_{gate} 512_{up}	4.746B$_{(59.10\%)}$	0.265	0.248	0	0
GlobaL Fact$_2$	512_{gate} 512_{up}	4.746B$_{(59.10\%)}$	0.267	0.249	0	0
GlobaL Fact$_2^*$	$\frac{512_{\text{gate}}}{512_{\text{up}}}$	4.746B$_{(59.10\%)}$	0.267	0.255	0	0

Table 7.7: Performance of Llama 3.1 (8B) compressed using GlobaL Fact on the benchmarks HellaSwag (HS), and GSM8K (GSM) before and after fine-tuning. Results for models before fine-tuning are indicated with the subscript i, while those after fine-tuning are labeled with f. GlobaL Fact$_1$ corresponds to GlobaL Fact with global matrices initialized using method 1, while GlobaL Fact$_2$ refers to initialization with method 2. GlobaL Fact$_2^*$, associated with $\frac{512_{\text{gate}}}{512_{\text{up}}}$, represents the sequential application of the compression framework and fine-tuning, first to the gate projection matrix and then to the up projection matrix.

remains in a region of the parameter space sufficiently close to a meaningful representation, allowing for effective adaptation. Fine-tuning facilitates a transition from random output generation to responses informed by an underlying understanding of the input language, indicating that certain fine-grained features may be shared across multiple layers (research question 3, Section 4.3).

However, despite the observed performance recovery, the compressed models remain significantly below their original state, indicating the impossibility of fully restoring the knowledge embedded in the original architecture. Given the compression ratio introduced by factorization, the resulting performance degradation is too severe to be considered acceptable.

A comparison between GlobaL Fact$_1$ and GlobaL Fact$_2$ confirms the superiority of the second over the first. GlobaL Fact$_2$ achieves higher performance than the SVD-based

compression when applied to gate projection matrices, whereas it underperforms in the compression of up projection layers. This outcome suggests a greater degree of similarity among gate projection matrices across the Transformer stack compared to up projection matrices.

When compressing multiple layers simultaneously, specifically the gate and up projections, retraining fails to preserve performance, leading the model to operate at random-guessing levels. In the case of GlobaL Fact$^*_2$, where the gate projection is compressed first, followed by the up projection, the two distinct phases of fine-tuning still fails to retain performance. Although this sequential compression strategy results in lower training loss compared to the approach where both layers are factorized and retrained simultaneously, it remains ineffective on benchmark evaluations, failing to produce meaningful results.

Details about the fine-tuning of the factorized models are provided in Table 7.8.

Across all settings reported in Table 7.8, training loss consistently decreases during fine-tuning, indicating successful learning. When both gate and up projection matrices are factorized, however, the training loss remains higher, aligning with random-guessing performance on benchmarks.

Validation loss, in contrast, is significantly higher than training loss and exhibits more fluctuation during fine-tuning. Many final values exceed the initial ones, suggesting the model may struggle to generalize beyond the training data. However, since benchmark performance recovery contradicts this, the hypothesis holds limited weight. The unusual loss behavior could also stem from the divergence between the training and validation datasets.

Although focusing solely on training performance is not ideal, final training losses in the range of 3 to 3.5 correlate with models that show non-random performance on the benchmarks. The training loss assessment can be considered robust, as it is evaluated on different samples, since an entire epoch cannot be completed due to the large size of OpenWebText. In contrast, validation loss appears less indicative of the accuracy and overall quality of the model.

Ultimately, GlobaL Fact represents an attempt to leverage shared features across Transformer blocks while preserving the local characteristics of individual layers. Although fine-tuning enables partial performance recovery, the degradation remains too severe for practical applications. However, the fine-tuning duration has been limited due to the large number of required experiments, longer training and further hyperparameter optimization could potentially improve the results.

Model	Ranks	#Params$_{(\text{CR})}$	loss$_{\text{train},1}$	loss$_{\text{train},f}$	loss$_{\text{val},i}$	loss$_{\text{val},f}$	Time
Truncated SVD	402_{gate}	6.388B$_{(79.55\%)}$	3.69	3.21	7.19	11.23	265 min
GlobaL Fact$_1$	512_{gate}	6.388B$_{(79.55\%)}$	4.09	3.25	8.74	10.75	267 min
GlobaL Fact$_2$	512_{gate}	6.388B$_{(79.55\%)}$	4.81	3.16	10.82	11.56	267 min
Truncated SVD	402_{up}	6.388B$_{(79.55\%)}$	4.01	3.22	13.09	12.68	266 min
GlobaL Fact$_1$	512_{up}	6.388B$_{(79.55\%)}$	6.85	3.47	13.33	12.78	265 min
GlobaL Fact$_2$	512_{up}	6.388B$_{(79.55\%)}$	6.16	3.44	12.54	15.66	267 min
Truncated SVD	402_{gate} 402_{up}	4.746B$_{(59.11\%)}$	6.26	4.88	13.57	13.10	228 min
GlobaL Fact$_1$	512_{gate} 512_{up}	4.746B$_{(59.10\%)}$	6.78	5.08	12.17	13.22	229 min
GlobaL Fact$_2$	512_{gate} 512_{up}	4.746B$_{(59.10\%)}$	7.15	5.25	12.92	10.99	230 min
GlobaL Fact$_2^*$	$\frac{512_{\text{gate}}}{512_{\text{up}}}$	4.746B$_{(59.10\%)}$	6.55	4.86	9.20	11.54	$\frac{267 \text{ min}}{218 \text{ min}}$

Table 7.8: Training and validation loss of Llama 3.1 (8B) compressed using GlobaL Fact, evaluated during fine-tuning. loss$_{\text{train},1}$ represents the average training loss computed on the samples considered before the first validation epoch, while loss$_{\text{train},f}$ corresponds to the average training loss measured between the last validation epoch and the end of retraining. Instead, loss$_{\text{val},i}$ and loss$_{\text{val},f}$ denote the validation loss before and after fine-tuning, respectively. Additionally, the total fine-tuning time (including validation periods) is reported. GlobaL Fact$_1$ corresponds to GlobaL Fact with global matrices initialized using method 1, while GlobaL Fact$_2$ refers to initialization with method 2. GlobaL Fact$_2^*$, associated with $\frac{512_{\text{gate}}}{512_{\text{up}}}$, represents the sequential application of the compression framework and fine-tuning, first to the gate projection matrix and then to the up projection matrix.

Approximation Error of the Initialization

Table 7.9 presents the total approximation errors of the matrices resulting from the application of low-rank factorization methods, evaluated before fine-tuning the model.

Given N matrices $\mathbf{W}_i$ and their approximations $\tilde{\mathbf{W}}_i$, the approximation error is measured using the average Sum of Squared Errors (SSE). Defining $\|\cdot\|_F$ as the Frobenius norm, SSE is computed as:

$$\text{SSE}_{\text{avg}} = \frac{1}{N} \sum_{i=0}^{N-1} \|\mathbf{W}_i - \tilde{\mathbf{W}}_i\|_F^2 \tag{7.4}$$

Similarly, the average Relative Sum of Squared Errors (RSSE) is defined as:

$$\text{RSSE}_{\text{avg}} = \frac{1}{N}\sum_{i=0}^{N-1}\frac{\|\mathbf{W}_i - \tilde{\mathbf{W}}_i\|_F^2}{\|\mathbf{W}_i\|_F^2} \tag{7.5}$$

Framework	Ranks	#Params	SSE_{avg}	RSSE_{avg}
Truncated SVD	402_{gate}	$6.388\text{B}_{(79.55\%)}$	9367.58	0.715
GlobaL Fact_1	512_{gate}	$6.388\text{B}_{(79.55\%)}$	10880.89	0.834
GlobaL Fact_2	512_{gate}	$6.388\text{B}_{(79.55\%)}$	11128.91	0.851
Truncated SVD	402_{up}	$6.388\text{B}_{(79.55\%)}$	6631.28	0.764
GlobaL Fact_1	512_{up}	$6.388\text{B}_{(79.55\%)}$	7444.17	0.856
GlobaL Fact_2	512_{up}	$6.388\text{B}_{(79.55\%)}$	7485.94	0.860

Table 7.9: Approximation errors after the factorization of the model, prior to fine-tuning. GlobaL Fact_1 corresponds to GlobaL Fact with global matrices initialized using method 1, while GlobaL Fact_2 refers to initialization with method 2.

As established by theoretical foundations, Truncated SVD provides the optimal low-rank approximation of a single matrix in terms of the Frobenius and spectral norms. Consequently, the model compressed using this approach exhibits the lowest approximation error. In contrast, the GlobaL Fact framework introduces additional error due to the inclusion of a shared global matrix, which is not specifically adapted to individual layers.

Nonetheless, the approximation error of GlobaL Fact remains close to that of the SVD-based initialization without shared components. The global matrix initialization methods 1 and 2 produce comparable errors, with 1 yielding slightly lower values. However, models initialized with 2 achieve better benchmark performance after fine-tuning.

Interestingly, despite the lower SSE_{avg} and RSSE_{avg} of truncated SVD, models compressed with GlobaL Fact exhibit similar performance in terms of losses and benchmark results, with GlobaL Fact_2 occasionally outperforming it. This suggests that factors beyond approximation error are critical to overall performance, emphasizing the complex interactions between model components.

7.2.4. Experiments on ABACO

This section presents the experimental setup, results, and key insights derived from evaluating the ABACO method.

Experimental Setup

The experiment follows the methodology outlined in Section 7.2.1. ABACO adapters are integrated into the model, which is then evaluated on HellaSwag (Section 5.3.2) and GSM8K (Section 5.3.3). The performance of the adapter-wrapped model remains identical to that of the original model, as LoRA does not alter the layer output before training. Subsequently, the model undergoes fine-tuning on OpenWebText (Section 5.2.2) and is re-evaluated on the same benchmarks.

Table 7.10 provides a summary of the key hyperparameters and experimental settings.

Variable	Value
Model	Gemma-2-2B
Fine-tuning dataset	OpenWebText
Validation dataset	WikiText-2
Batch size	2
Learning rate	10^{-3}
Number of batches	50K
Gradient accumulation steps	2
Optimizer	AdamW
Model precision	16-bit
Exponential decay of α	$2^{-\frac{x}{16000}}$

Table 7.10: Hyperparameters and fine-tuning settings for the experiments conducted on ABACO.

Experiments are conducted by factorizing all layers of a given type. As in other factorization techniques, ABACO also requires specifying the internal dimension, which determines the level of approximation. The factorization of matrices of type t using rank r is denoted as r_t in the following tables.

Evaluation of ABACO-Based Compression of Gate Projection Matrices

Table 7.11 presents the benchmark results for the model where gate projection matrices are compressed using ABACO.

Model	Ranks	#Params$_{(CR)}$	HS$_i$	HS$_f$	GSM$_i$	GSM$_f$
Original Model	-	2.614B	0.730		0.246	
ABACO	288$_{gate}$	2.149B$_{(82.18\%)}$	0.730	0.337	0.246	0.008

Table 7.11: Performance of Gemma 2 (2B) when ABACO is applied to gate projection matrices on the benchmarks HellaSwag (HS), and GSM8K (GSM) before and after fine-tuning. Results for models before fine-tuning are indicated with the subscript i, while those after fine-tuning are labeled with f.

The results indicate that ABACO preserves some of the capabilities of the original model after factorization. The method enables partial knowledge transfer, as demonstrated by its ability to achieve non-random final performance. However, the substantial accuracy gap between the compressed and original LLM highlights its limitations. Although some degradation is expected, the observed decline is very severe. In particular, performance on complex reasoning tasks such as GSM8K deteriorates significantly, suggesting that the method struggles to retain the essential information.

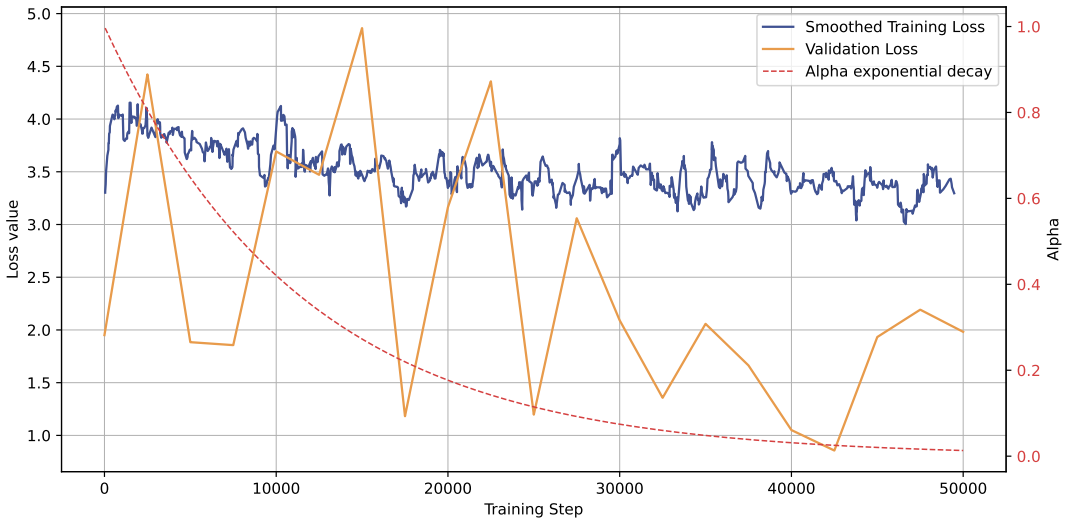


Figure 7.36: Training and validation loss during fine-tuning of Gemma 2 (2B) with ABACO applied to gate projections. The figure also depicts the exponential alpha decay. The training loss is smoothed using a window of 10 elements. The total fine-tuning time is 12 hours and 6 minutes.

The loss trends depicted in Figure 7.36 provide further insights. Initially, the training loss spikes (rising from approximately 2 in the original model to around 4), likely due to the reliance on randomly initialized adapters that require time to adapt. As training

progresses the training loss decreases while pre-trained weights are progressively discarded. This suggests that knowledge transfer occurs successfully, allowing the model to adjust to its compressed form without catastrophic failure. The controlled decay of alpha effectively prevents abrupt disruptions in training dynamics.

The validation loss fluctuates significantly throughout training, exhibiting a high degree of instability. Nonetheless, the fact that it does not diverge is a positive sign. Toward the end of training, it settles to a relatively low value. Although this behavior might indicate some level of adaptation, it does not necessarily translate into improved performance on downstream tasks, as benchmark results show that the compressed model remains far from matching the original.

Evaluation of ABACO-Based Compression of Query Projection Matrices

Table 7.12 presents the benchmark results for the model where query projection matrices are compressed using ABACO.

Model	Ranks	#Params$_{(CR)}$	HS$_i$	HS$_f$	GSM$_i$	GSM$_f$
Original Model	-	2.614B	0.730		0.246	
ABACO	144$_{\text{query}}$	2.494B$_{(95.38\%)}$	0.730	0.306	0.246	0.003

Table 7.12: Performance of Gemma 2 (2B) when ABACO is applied to query weight matrices on the benchmarks HellaSwag (HS), and GSM8K (GSM) before and after fine-tuning. Results for models before fine-tuning are indicated with the subscript i, while those after fine-tuning are labeled with f.

Despite the relatively small reduction in parameters, applying ABACO to query matrices results in substantial performance degradation. Benchmark scores drop significantly post-compression, indicating that query projections are particularly sensitive to this process. The final results suggest that the method struggles to preserve critical information necessary for downstream tasks.

Figure 7.37 provides further insights into this behavior. Initially, the training loss stabilizes after an increase similar to that observed when compressing gate projections. However, as alpha continues to decay, performance deteriorates. Unlike in the gate projection case, where the loss follows a consistent downward trajectory, here it begins to rise again, suggesting that fine-tuning struggles to fully compensate for the removed information. This trend was observed multiple times, underscoring the challenge of determining an

optimal decay rate for alpha that enables effective compression. Moreover, in the context of the self-attention mechanism, finding an appropriate decay rate proved even more challenging, complicating the compression process.

Similarly to gate compression, query compression also results in significant fluctuations in validation loss throughout fine-tuning, often reaching higher values. These elevated validation loss levels further highlight the learning difficulties in this scenario.

Figure 7.37: Training and validation loss during fine-tuning of Gemma 2 (2B) with ABACO applied to query projections. The figure also depicts the exponential alpha decay. The training loss is smoothed using a window of 10 elements. The total duration of the fine-tuning is 11 hours and 33 minutes.

A more extensive hyperparameter search could help achieve better convergence and improved benchmark performance in this setting as well.

Designed to address Research Question 2 (Section 4.2), ABACO leverages low-rank structures to enable smooth and continuous factorization. Although the approach is viable, the results reveal critical limitations that hinder its effectiveness. ABACO prevents collapse and preserves a functional model, yet it fails to sustain competitive performance. The substantial gap between the compressed and original models renders it impractical for real-world applications, underscoring the need for further refinement.

A more comprehensive analysis of the results of ABACO across different layer roles and rank configurations could help identify optimal trade-offs between compression and performance retention. Additionally, full-scale retraining on a larger and more diverse dataset may yield a model that better preserves the knowledge embedded in the original.

Further exploration of hyperparameters, particularly in decay strategies and training setting, could also enhance overall effectiveness of the method.

8 | Conclusions and Future Developments

This chapter summarizes the main findings of the thesis and discusses their significance within the field of natural language processing. It outlines the key contributions, addresses the challenges faced during the research, and explores the limitations of the work. Additionally, it highlights opportunities for future research.

8.1. Conclusions

Driven by the need for a deeper understanding of the internal mechanisms of large language models and the prevailing assumption of over-parameterization, this thesis investigates redundancy within Transformer-based architectures, aiming to assess the extent to which layers execute similar transformations. This inquiry is further extended to evaluate the informational content of weight matrices and analyze its implications for model compression.

The findings challenge the presence of functional redundancy across sub-blocks. Contrary to expectations, modules with analogous roles exhibit distinct, block-dependent transformations rather than partially replicating each other's operations. This insight questions conventional assumptions about redundancy in deep learning architectures and emphasizes the need for compression strategies that account for the nuanced roles of different layers.

Given the ever-increasing size and computational demands of modern large language models, this thesis presents several compression techniques aiming to reduce model dimensions while preserving performance. Three approaches are proposed and empirically evaluated: MASS, GlobaL Fact and ABACO. Each method exploits redundancy through distinct strategies, investigating whether parameter sharing and factorization can effectively enable model compression.

MASS compresses models by aggregating multiple matrices into a single shared represen-

tation, operating under the assumption that layers can be effectively merged and shared. The method consists of two main steps: grouping layers based on functional roles and aggregating them using simple averaging. However, experimental results demonstrate that this approach leads to severe performance degradation, even after fine-tuning. The case where layers are fully shared across all Transformer blocks results in a model that performs randomly on benchmarks. The failure of this method can be attributed to an overly aggressive grouping strategy and the simplistic nature of the aggregation mechanism, which fails to capture the nuanced transformations performed by individual layers. Allowing only partial aggregation and sharing in the middle blocks enables some recovery of performance. The outcomes remain consistent with the findings of replacement analysis, further confirming that redundancy between layers is insufficient for direct parameter sharing.

GlobaL Fact adopts a factorization-based approach, decomposing linear transformations into two components: a global matrix, shared across multiple layers, and local matrices, which retain layer-specific variations. The underlying assumption is that, while layers are not identical, they may exhibit shared patterns that can be leveraged for compression. However, factorization using shared components initially leads to significant performance degradation. Fine-tuning partially restores lost performance, demonstrating that the approximated model retains a meaningful internal structure that allows for some recovery. This improvement remains insufficient to bridge the gap with the original model, and when multiple layer types are factorized simultaneously, performance recovery becomes even more challenging.

ABACO introduces a progressive compression framework built upon adapter-based fine-tuning. Instead of directly modifying weight matrices, this method gradually transfers model knowledge from full-rank weights to low-rank adapters through a penalization mechanism that progressively reduces reliance on the original weights. The motivation behind this approach is that abrupt parameter truncation, as observed in traditional factorization methods, often results in a catastrophic loss of knowledge. Experimental results indicate that ABACO facilitates progressive model compression while preserving some functionality after fine-tuning. However, performance degradation remains significant, particularly in complex reasoning tasks, where accuracy drops considerably post-compression.

The analysis of these compression methods reveals several fundamental challenges and limitations. A key insight is that inter-layer information sharing is not trivially exploitable. While certain degrees of redundancy exist within weight matrices and layer operations, as evidenced by the partial success of GlobaL Fact, direct parameter sharing or factorization

often leads to a loss of essential functional distinctions. The experiments confirm that layer-wise transformations are more specialized than initially presumed, rendering naive parameter-sharing strategies not very effective.

Overall, this thesis contributes to ongoing research on large language model optimization by providing empirical insights into redundancy within Transformer architectures and evaluating the feasibility of various compression strategies. The findings highlight that Transformer-based models exhibit a more intricate structure necessitating more refined approaches to compression.

8.2. Future Developments

A comprehensive understanding of the internal mechanisms of Transformer models remains an open challenge. Future research could investigate functional redundancy at multiple levels identified in this thesis but not yet thoroughly examined. Analyzing fine-grained dependencies, such as relationships between attention heads or individual feed forward layers, alongside broader structural similarities, for instance among Transformer blocks, may uncover additional insights and possibilities.

Extending the proposed compression approaches to a broader range of models and tasks could provide deeper insights into their effectiveness. Investigating redundancy in models beyond those analyzed may reveal structural patterns better suited for compression, further highlighting the potential of these techniques.

Another possible direction is an extensive hyperparameter optimization study. Key factors such as the degree of factorization and the number of shared components could enhance the viability of the proposed methods. This would help determine whether current limitations stem from inherent constraints within the methods or result from suboptimal configurations. Additionally, a more in-depth exploration of the suitability of different layer types for compression by each method, which is only partially addressed in this thesis, could offer valuable insights for optimizing compression strategies.

Further improvements could involve refining layer grouping strategies to reduce the loss of functional specificity while preserving computational efficiency. Alternative aggregation and initialization mechanisms could improve structural alignment within weight matrices and mitigate the degradation observed in naive weight-sharing approaches.

Bibliography

[1] J. Alammar. The illustrated transformer [blog post]. https://jalammar.github.io/illustrated-transformer, 2018.

[2] R. Anil, S. Borgeaud, Y. Wu, J. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, K. Millican, D. Silver, S. Petrov, M. Johnson, I. Antonoglou, J. Schrittwieser, A. Glaese, J. Chen, E. Pitler, T. P. Lillicrap, A. Lazaridou, O. Firat, J. Molloy, M. Isard, P. R. Barham, T. Hennigan, B. Lee, F. Viola, M. Reynolds, Y. Xu, R. Doherty, E. Collins, C. Meyer, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, G. Tucker, E. Piqueras, M. Krikun, I. Barr, N. Savinov, I. Danihelka, B. Roelofs, A. White, A. Andreassen, T. von Glehn, L. Yagati, M. Kazemi, L. Gonzalez, M. Khalman, J. Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023. doi: 10.48550/ARXIV.2312.11805. URL https://doi.org/10.48550/arXiv.2312.11805.

[3] S. Ashkboos, M. L. Croci, M. G. D. Nascimento, T. Hoefler, and J. Hensman. Slicegpt: Compress large language models by deleting rows and columns. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=vXxardq6db.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

[5] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. 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. URL https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

[6] R. Chand, Y. Prabhu, and P. Kumar. Dsformer: Effective compression of text-transformers by dense-sparse weight factorization. *CoRR*, abs/2312.13211, 2023. doi: 10.48550/ARXIV.2312.13211. URL https://doi.org/10.48550/arXiv.2312.13211.

[7] P. H. Chen, H. Yu, I. S. Dhillon, and C. Hsieh. DRONE: data-aware low-rank compression for large NLP models. In M. Ranzato, A. Beygelzimer, Y. N. Dauphin, P. Liang, and J. W. 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 29321–29334, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f56de5ef149cf0aedcc8f4797031e229-Abstract.html.

[8] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL http://arxiv.org/abs/1412.3555.

[9] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

[10] J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

[11] D. Du, Y. Zhang, S. Cao, J. Guo, T. Cao, X. Chu, and N. Xu. Bitdistiller: Unleashing the potential of sub-4-bit llms via self-distillation. In L. Ku, A. Martins, and V. 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 102–116. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.7. URL https://doi.org/10.18653/v1/2024.acl-long.7.

[12] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, A. Goyal, A. Hartshorn, A. Yang, A. Mitra, A. Sravankumar, A. Korenev, A. Hinsvark, A. Rao, A. Zhang, A. Rodriguez, A. Gregerson,

A. Spataru, B. Rozière, B. Biron, B. Tang, B. Chern, C. Caucheteux, C. Nayak, C. Bi, C. Marra, C. McConnell, C. Keller, C. Touret, C. Wu, C. Wong, C. C. Ferrer, C. Nikolaidis, D. Allonsius, D. Song, D. Pintz, D. Livshits, D. Esiobu, D. Choudhary, D. Mahajan, D. Garcia-Olano, D. Perino, D. Hupkes, E. Lakomkin, E. AlBadawy, E. Lobanova, E. Dinan, E. M. Smith, F. Radenovic, F. Zhang, G. Synnaeve, G. Lee, G. L. Anderson, G. Nail, G. Mialon, G. Pang, G. Cucurell, H. Nguyen, H. Korevaar, H. Xu, H. Touvron, I. Zarov, I. A. Ibarra, I. M. Kloumann, I. Misra, I. Evtimov, J. Copet, J. Lee, J. Geffert, J. Vranes, J. Park, J. Mahadeokar, J. Shah, J. van der Linde, J. Billock, J. Hong, J. Lee, J. Fu, J. Chi, J. Huang, J. Liu, J. Wang, J. Yu, J. Bitton, J. Spisak, J. Park, J. Rocca, J. Johnstun, J. Saxe, J. Jia, K. V. Alwala, K. Upasani, K. Plawiak, K. Li, K. Heafield, K. Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL `https://doi.org/10.48550/arXiv.2407.21783`.

[13] C. Eckart and G. Young. The approximation of one matrix by another of lower rank. *Psychometrika*, 1(3):211–218, 1936. doi: 10.1007/BF02288367.

[14] J. L. Elman. Finding structure in time. *Cogn. Sci.*, 14(2):179–211, 1990. doi: 10.1207/S15516709COG1402_1. URL `https://doi.org/10.1207/s15516709cog1402_1`.

[15] E. Frantar and D. Alistarh. Optimal brain compression: A framework for accurate post-training quantization and pruning. In S. Koyejo, S. Mohamed, A. Agarwal, D. 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. URL `http://papers.nips.cc/paper_files/paper/2022/hash/1caf09c9f4e6b0150b06a07e77f2710c-Abstract-Conference.html`.

[16] E. Frantar and D. Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. 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 10323–10337. PMLR, 2023. URL `https://proceedings.mlr.press/v202/frantar23a.html`.

[17] E. Frantar, S. Ashkboos, T. Hoefler, and D. Alistarh. GPTQ: accurate post-training quantization for generative pre-trained transformers. *CoRR*, abs/2210.17323, 2022. doi: 10.48550/ARXIV.2210.17323. URL `https://doi.org/10.48550/arXiv.2210.17323`.

[18] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36: 193–202, 1980. URL https://api.semanticscholar.org/CorpusID:206775608.

[19] T. Gale, E. Elsen, and S. Hooker. The state of sparsity in deep neural networks. *CoRR*, abs/1902.09574, 2019. URL http://arxiv.org/abs/1902.09574.

[20] L. Gao, J. Tow, B. Abbasi, S. Biderman, S. Black, A. DiPofi, C. Foster, L. Golding, J. Hsu, A. Le Noac'h, H. Li, K. McDonell, N. Muennighoff, C. Ociepa, J. Phang, L. Reynolds, H. Schoelkopf, A. Skowron, L. Sutawika, E. Tang, A. Thite, B. Wang, K. Wang, and A. Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

[21] A. Gokaslan and V. Cohen. Openwebtext corpus. http://Skylion007.github.io/OpenWebTextCorpus, 2019.

[22] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Trans. Inf. Theory*, 44(6):2325–2383, 1998. doi: 10.1109/18.720541. URL https://doi.org/10.1109/18.720541.

[23] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgeon and general network pruning. In *Proceedings of International Conference on Neural Networks (ICNN'88), San Francisco, CA, USA, March 28 - April 1, 1993*, pages 293–299. IEEE, 1993. doi: 10.1109/ICNN.1993.298572. URL https://doi.org/10.1109/ICNN.1993.298572.

[24] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. URL http://arxiv.org/abs/1503.02531.

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, 1997. doi: 10.1162/NECO.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[26] Y. Hsu, T. Hua, S. Chang, Q. Lou, Y. Shen, and H. Jin. Language model compression with weighted low-rank factorization. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=uPv9Y3gmAI5.

[27] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

[28] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. de Las Casas,

F. Bressand, G. Lengyel, G. Lample, L. Saulnier, L. R. Lavaud, M. Lachaux, P. Stock, T. L. Scao, T. Lavril, T. Wang, T. Lacroix, and W. E. Sayed. Mistral 7b. *CoRR*, abs/2310.06825, 2023. doi: 10.48550/ARXIV.2310.06825. URL `https://doi.org/10.48550/arXiv.2310.06825`.

[29] M. I. Jordan. Chapter 25 - serial order: A parallel distributed processing approach. In J. W. Donahoe and V. Packard Dorsel, editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 471–495. North-Holland, 1997. doi: https://doi.org/10.1016/S0166-4115(97)80111-2. URL `https://www.sciencedirect.com/science/article/pii/S0166411597801112`.

[30] D. Jurafsky and J. H. Martin. *Speech and Language Processing (2Nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2009. ISBN 0131873210.

[31] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. None, 3rd edition, 2025. URL `https://web.stanford.edu/~jurafsky/slp3/`. Online manuscript released January 12, 2025.

[32] D. J. Kopiczko, T. Blankevoort, and Y. M. Asano. Vera: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=NjNfLdxr3A`.

[33] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989. URL `https://proceedings.neurips.cc/paper_files/paper/1989/file/53c3bce66e43be4f209556518c2fcb54-Paper.pdf`.

[34] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, pages 598–605. Morgan Kaufmann, 1989. URL `http://papers.nips.cc/paper/250-optimal-brain-damage`.

[35] J. Lin, J. Tang, H. Tang, S. Yang, G. Xiao, and S. Han. AWQ: activation-aware weight quantization for on-device LLM compression and acceleration. *GetMobile Mob. Comput. Commun.*, 28(4):12–17, 2024. doi: 10.1145/3714983.3714987. URL `https://doi.org/10.1145/3714983.3714987`.

[36] S. Lin, J. Hilton, and O. Evans. Truthfulqa: Measuring how models mimic human falsehoods. In S. Muresan, P. Nakov, and A. Villavicencio, editors, *Proceedings of the*

60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022, pages 3214–3252. Association for Computational Linguistics, 2022. doi: 10.18653/V1/2022.ACL-LONG.229. URL https://doi.org/10.18653/v1/2022.acl-long.229.

[37] Z. Liu, J. Wang, T. Dao, T. Zhou, B. Yuan, Z. Song, A. Shrivastava, C. Zhang, Y. Tian, C. Ré, and B. Chen. Deja vu: Contextual sparsity for efficient llms at inference time. In A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. 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 22137–22176. PMLR, 2023. URL https://proceedings.mlr.press/v202/liu23am.html.

[38] Z. Liu, B. Oguz, C. Zhao, E. Chang, P. Stock, Y. Mehdad, Y. Shi, R. Krishnamoorthi, and V. Chandra. LLM-QAT: data-free quantization aware training for large language models. In L. Ku, A. Martins, and V. Srikumar, editors, *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 467–484. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.FINDINGS-ACL.26. URL https://doi.org/10.18653/v1/2024.findings-acl.26.

[39] X. Ma, G. Fang, and X. Wang. Llm-pruner: On the structural pruning of large language models. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. 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. URL http://papers.nips.cc/paper_files/paper/2023/hash/44956951349095f74492a5471128a7e0-Abstract-Conference.html.

[40] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In Y. Bengio and Y. LeCun, editors, *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, 2013. URL http://arxiv.org/abs/1301.3781.

[41] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort. A white paper on neural network quantization. *CoRR*, abs/2106.08295, 2021. URL https://arxiv.org/abs/2106.08295.

[42] M. B. Noach and Y. Goldberg. Compressing pre-trained language models by matrix decomposition. In K. Wong, K. Knight, and H. Wu, editors, *Proceedings of the*

1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020, pages 884–889. Association for Computational Linguistics, 2020. URL https://aclanthology.org/2020.aacl-main.88/.

[43] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL https://doi.org/10.48550/arXiv.2303.08774.

[44] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. *OpenAI Technical Report*, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

[45] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Technical Report*, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[46] M. Rivière, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret, P. Liu, P. Tafti, A. Friesen, M. Casbon, S. Ramos, R. Kumar, C. L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C. A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozinska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucinska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J. P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjösund, L. Usui, L. Sifre, L. Heuermann, L. Lago, and L. McNealus. Gemma 2: Improving open language models at a practical size. *CoRR*, abs/2408.00118, 2024. doi: 10.48550/ARXIV.2408.00118. URL https://doi.org/10.48550/arXiv.2408.00118.

[47] P. Sharma, J. T. Ash, and D. Misra. The truth is in there: Improving reasoning in language models with layer-selective rank reduction. In *The Twelfth International Con-*

ference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024. URL https://openreview.net/forum?id=ozX92bu8VA.

[48] S. P. Singh and D. Alistarh. Woodfisher: Efficient second-order approximation for neural network compression. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. 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. URL https://proceedings.neurips.cc/paper/2020/hash/d1ff1ec86b62cd5f3903ff19c3a326b2-Abstract.html.

[49] J. Song, K. Oh, T. Kim, H. Kim, Y. Kim, and J. Kim. SLEB: streamlining llms through redundancy verification and elimination of transformer blocks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=fuX4hyLPm0.

[50] . Stephen Merity et al. Wikitext-2, 2016. URL https://arxiv.org/abs/1609.07843.

[51] J. Su, M. H. M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. doi: 10.1016/J.NEUCOM.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

[52] M. Sun, Z. Liu, A. Bair, and J. Z. Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=PxoFut3dWW.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. 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. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

[54] X. Wang, Y. Zheng, Z. Wan, and M. Zhang. SVD-LLM: truncation-aware singular value decomposition for large language model compression. *CoRR*, abs/2403.07378, 2024. doi: 10.48550/ARXIV.2403.07378. URL https://doi.org/10.48550/arXiv.2403.07378.

[55] Z. Yuan, Y. Shang, Y. Song, Q. Wu, Y. Yan, and G. Sun. ASVD: activation-aware singular value decomposition for compressing large language models. *CoRR*, abs/2312.05821, 2023. doi: 10.48550/ARXIV.2312.05821. URL https://doi.org/10.48550/arXiv.2312.05821.

[56] R. Zellers, A. Holtzman, Y. Bisk, A. Farhadi, and Y. Choi. Hellaswag: Can a machine really finish your sentence? In A. Korhonen, D. R. Traum, and L. 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 4791–4800. Association for Computational Linguistics, 2019. doi: 10.18653/V1/P19-1472. URL https://doi.org/10.18653/v1/p19-1472.

[57] B. Zhang and R. Sennrich. Root mean square layer normalization. In H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 12360–12371, 2019. URL https://proceedings.neurips.cc/paper/2019/hash/1e8a19426224ca89e83cef47f1e7f53b-Abstract.html.

[58] Q. Zhang, M. Chen, A. Bukharin, N. Karampatziakis, P. He, Y. Cheng, W. Chen, and T. Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023. URL https://arxiv.org/abs/2303.10512.

[59] M. Zhu and S. Gupta. To prune, or not to prune: Exploring the efficacy of pruning for model compression. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Workshop Track Proceedings*. OpenReview.net, 2018. URL https://openreview.net/forum?id=Sy1iIDkPM.

[60] X. Zhu, J. Li, Y. Liu, C. Ma, and W. Wang. A survey on model compression for large language models. *CoRR*, abs/2308.07633, 2023. doi: 10.48550/ARXIV.2308.07633. URL https://doi.org/10.48550/arXiv.2308.07633.

A | Appendix A

This appendix contains the results of the singular value distribution and rank analysis for the weight matrices of models that were not included in Section 7.1.2. Specifically, Mistral-7B-v0.3, Gemma-2-2B, and Gemma-2-9B are analyzed.

A.1. Intra-Layers Rank Analysis

This section presents the singular value distributions and approximate ranks of individual linear layer weight matrices within the self-attention and feed-forward modules. The methodology follows the framework outlined in Section 7.1.2.

A.1.1. Mistral-7B-v0.3

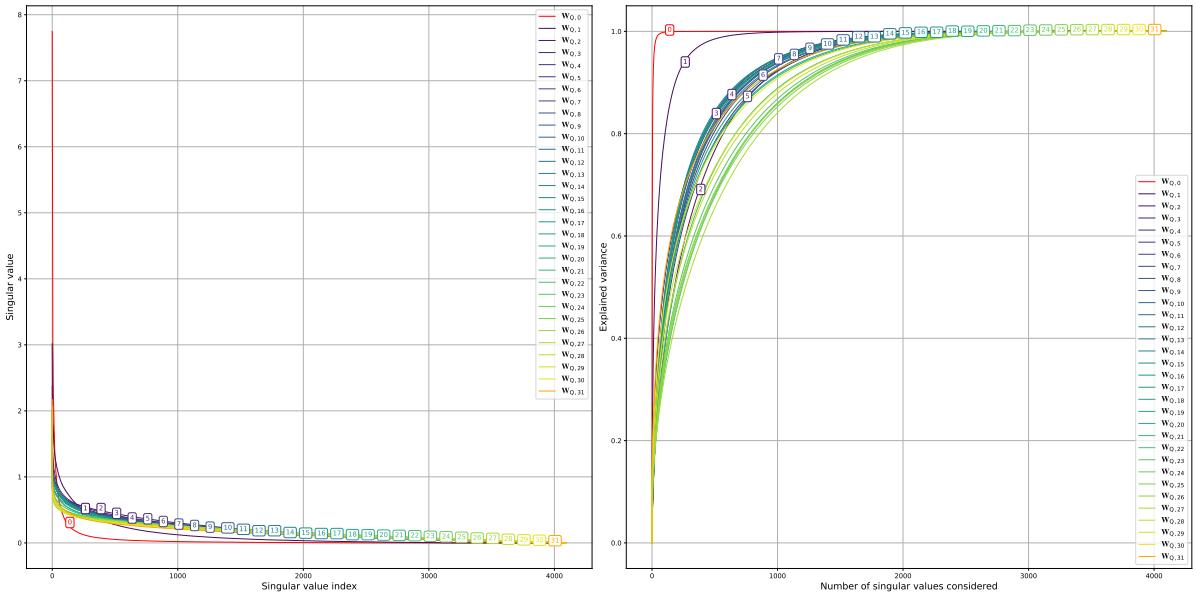


Figure A.1: Singular value distribution and cumulative explained variance of self-attention query matrices in Mistral blocks.

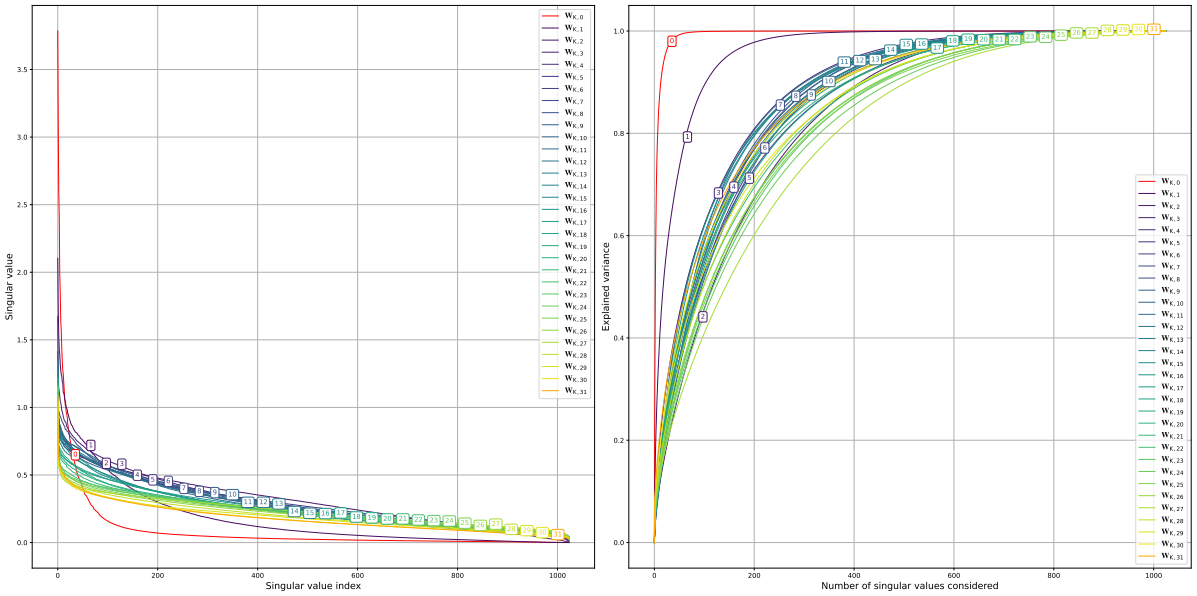


Figure A.2: Singular value distribution and cumulative explained variance of self-attention key matrices in Mistral blocks.

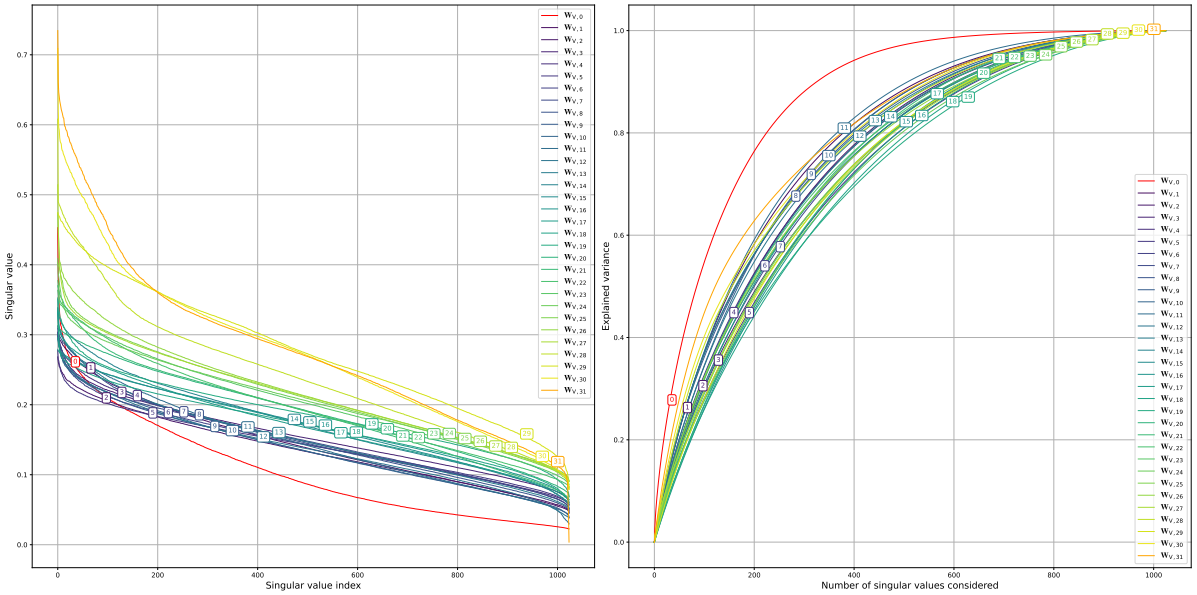


Figure A.3: Singular value distribution and cumulative explained variance of self-attention value matrices in Mistral blocks.

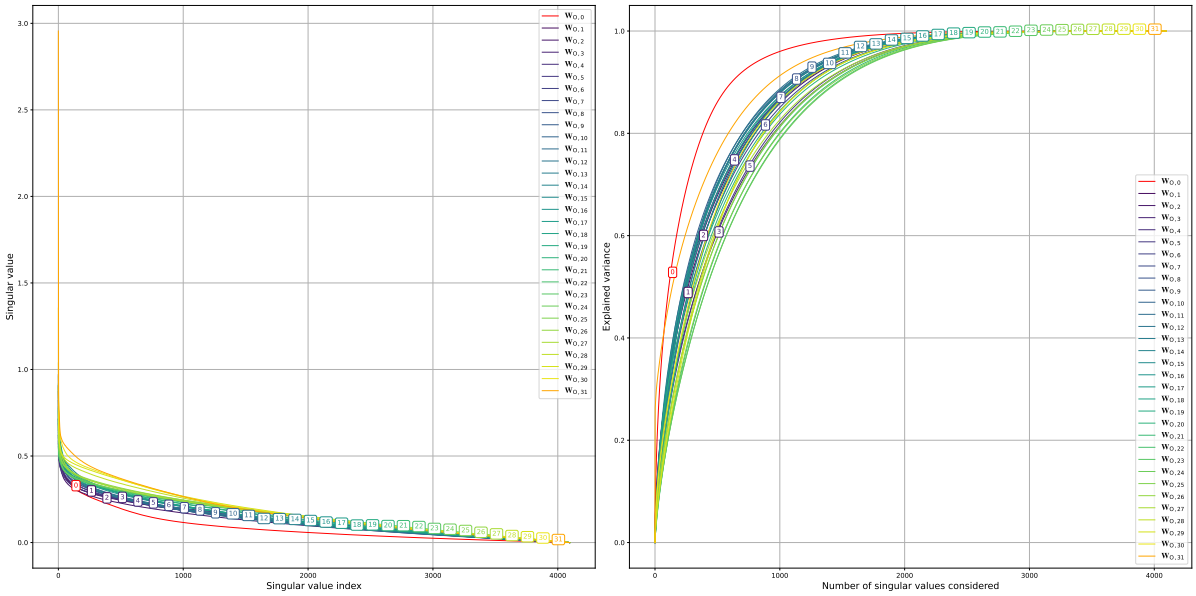


Figure A.4: Singular value distribution and cumulative explained variance of self-attention output matrices in Mistral blocks.

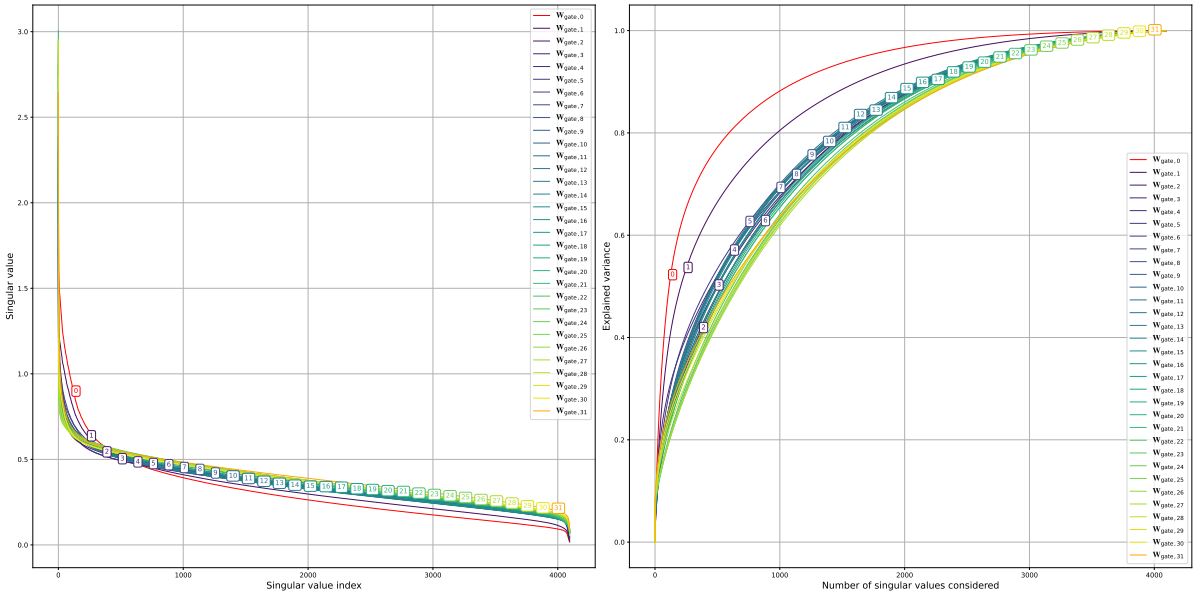


Figure A.5: Singular value distribution and cumulative explained variance of gate projection matrices in Mistral blocks.

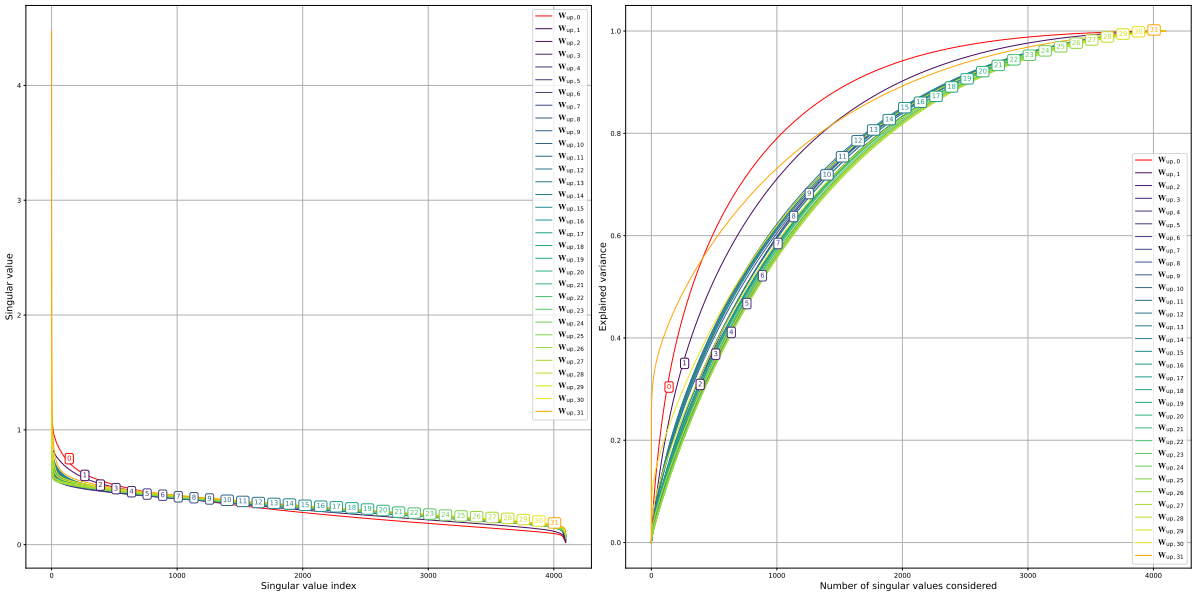


Figure A.6: Singular value distribution and cumulative explained variance of up projection matrices in Mistral blocks.

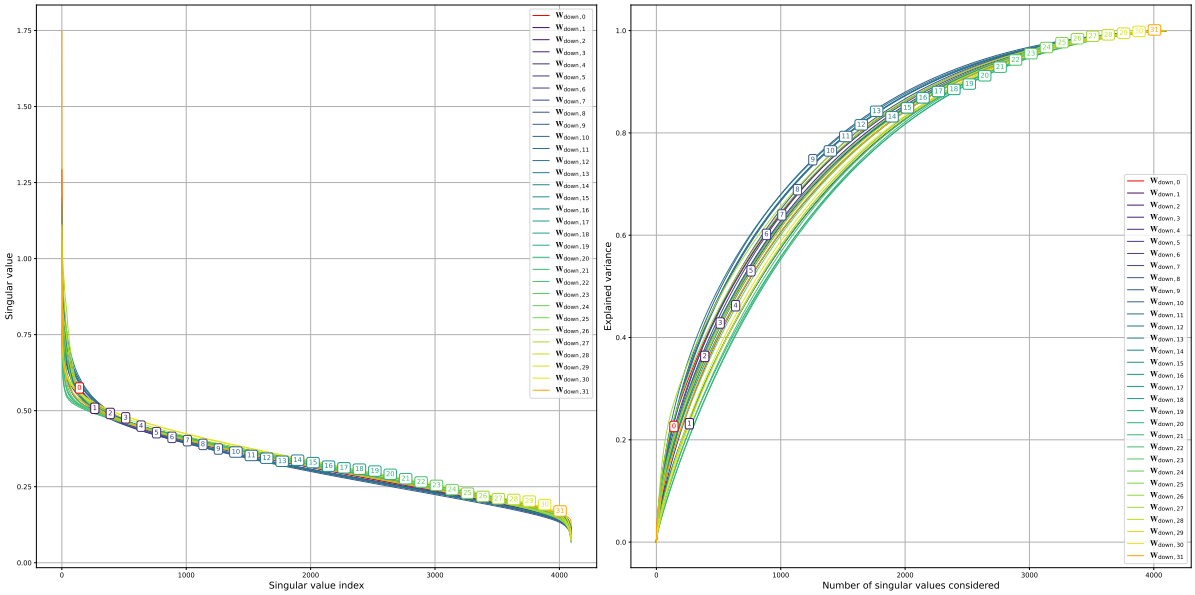


Figure A.7: Singular value distribution and cumulative explained variance of down projection matrices in Mistral blocks.

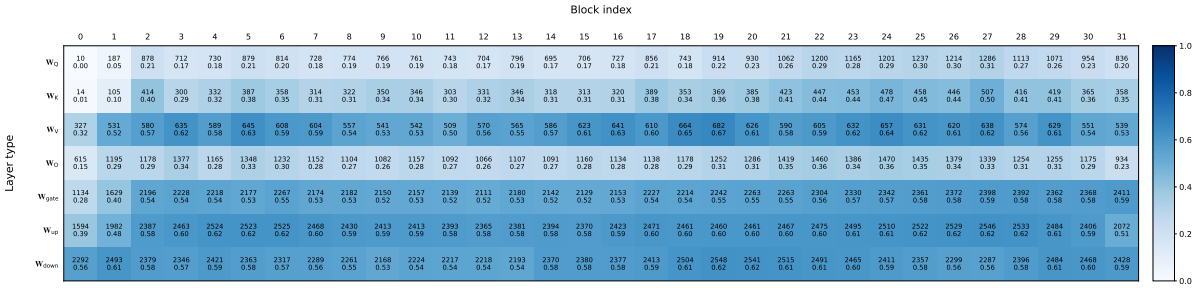


Figure A.8: Approximate ranks of the weight matrices in Mistral, determined using a threshold of 0.9 for the explained variance.

A.1.2. Gemma-2-2b

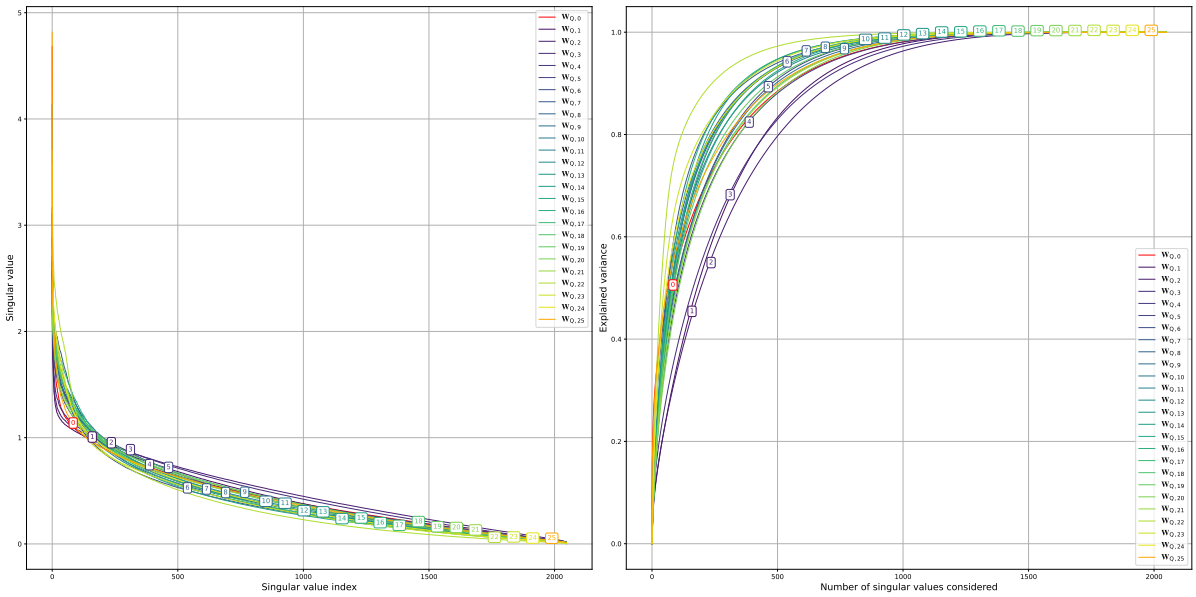


Figure A.9: Singular value distribution and cumulative explained variance of self-attention query matrices in Gemma 2 (2B) blocks.

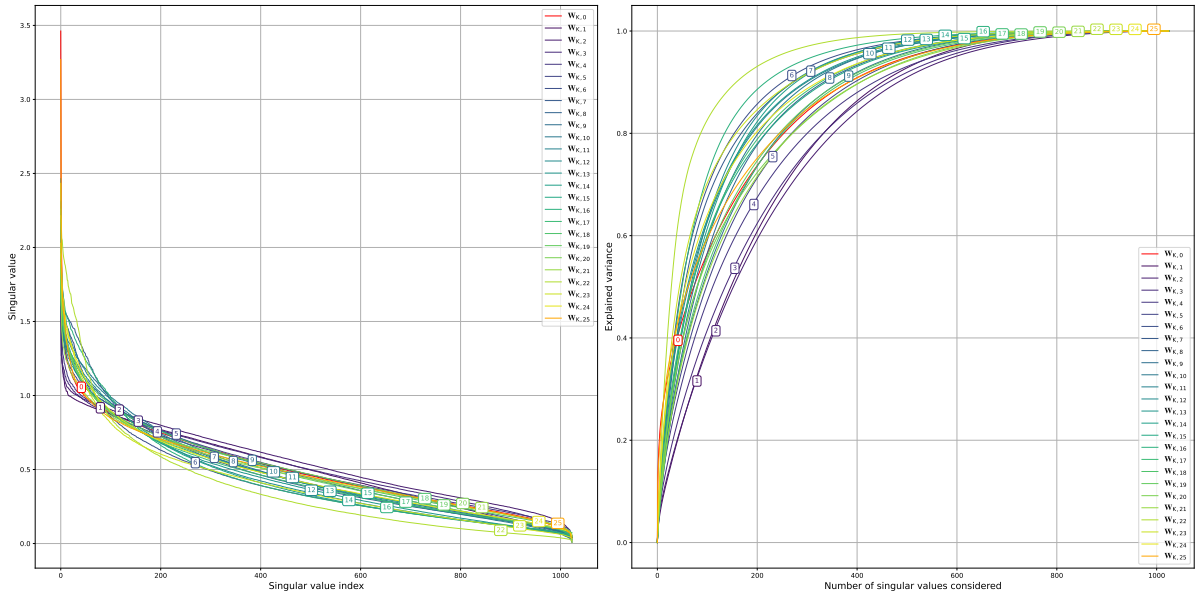


Figure A.10: Singular value distribution and cumulative explained variance of self-attention key matrices in Gemma 2 (2B) blocks.

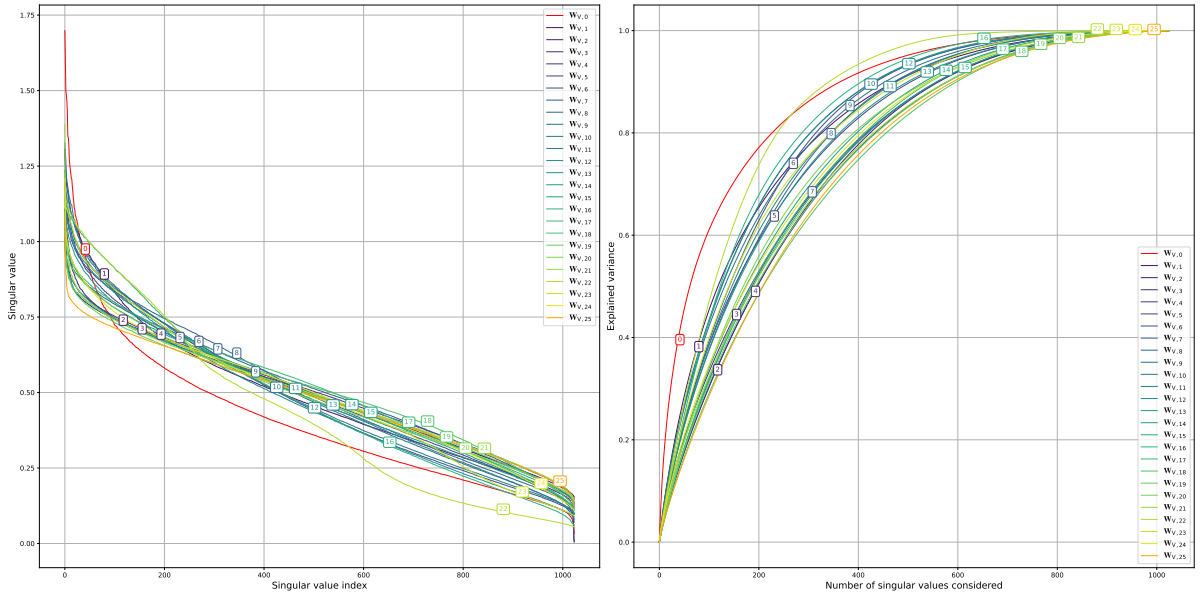


Figure A.11: Singular value distribution and cumulative explained variance of self-attention value matrices in Gemma 2 (2B) blocks.

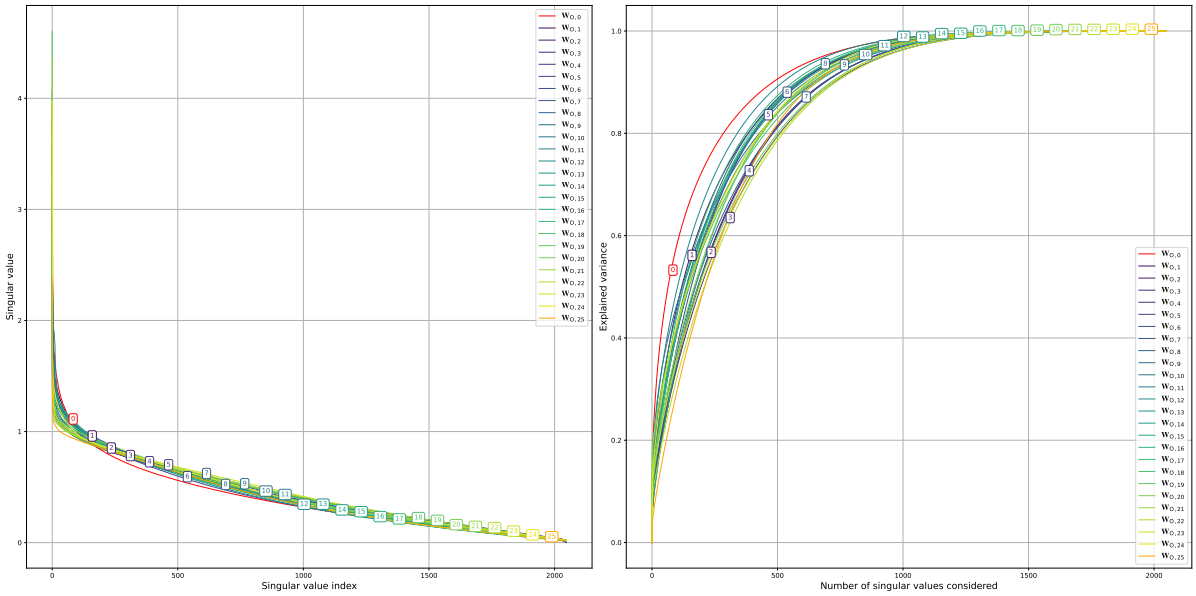


Figure A.12: Singular value distribution and cumulative explained variance of self-attention output matrices in Gemma 2 (2B) blocks.

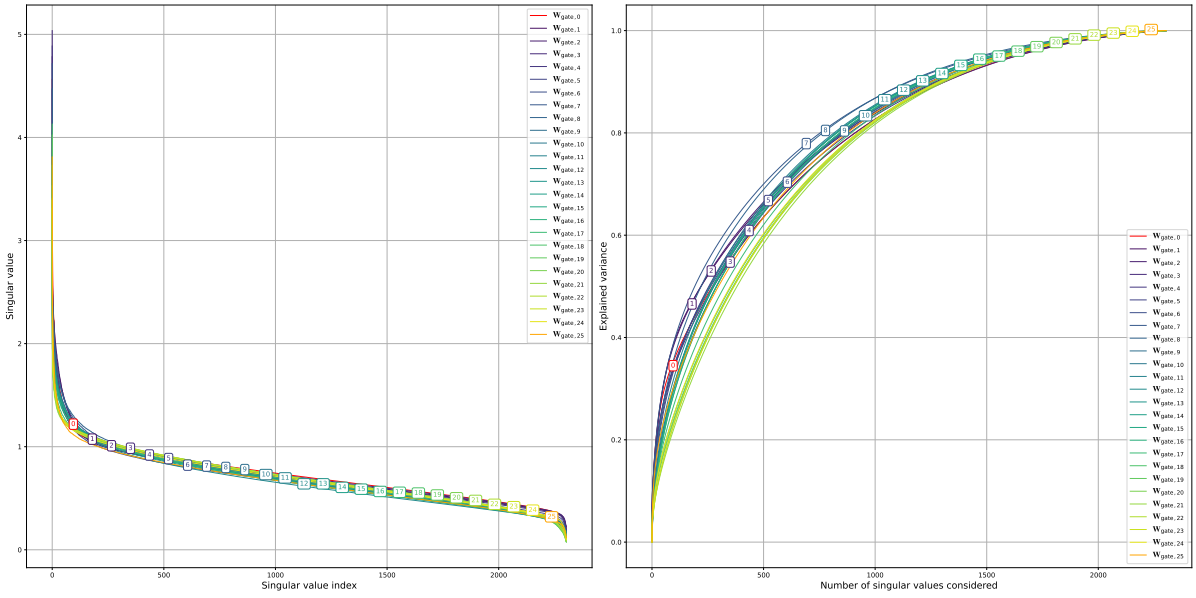


Figure A.13: Singular value distribution and cumulative explained variance of gate projection matrices in Gemma 2 (2B) blocks.

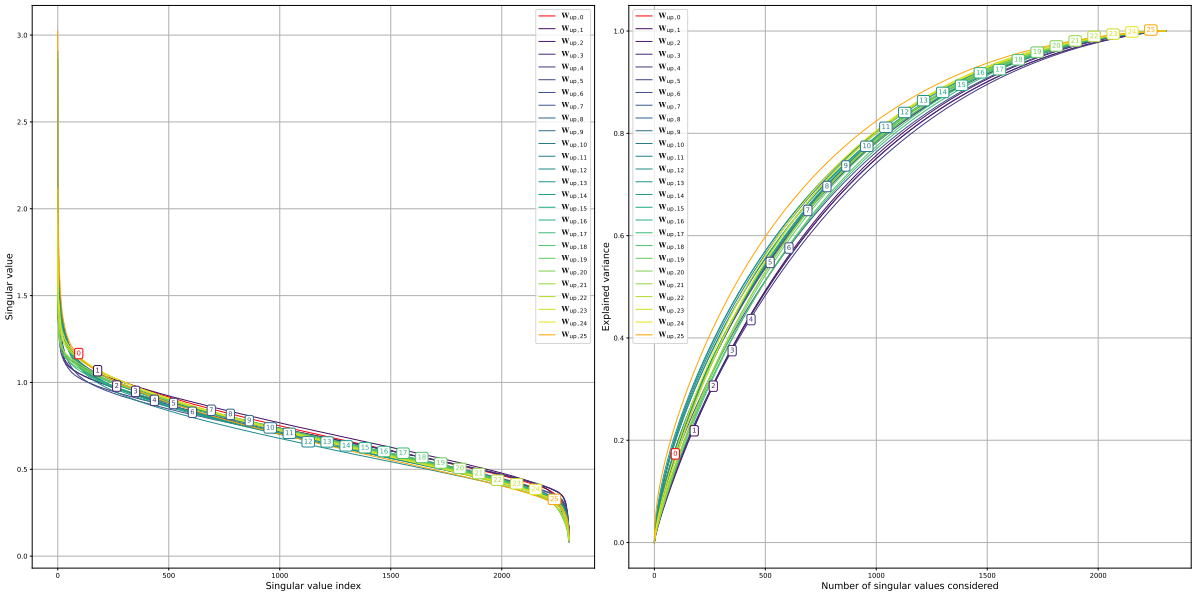


Figure A.14: Singular value distribution and cumulative explained variance of up projection matrices in Gemma 2 (2B) blocks.

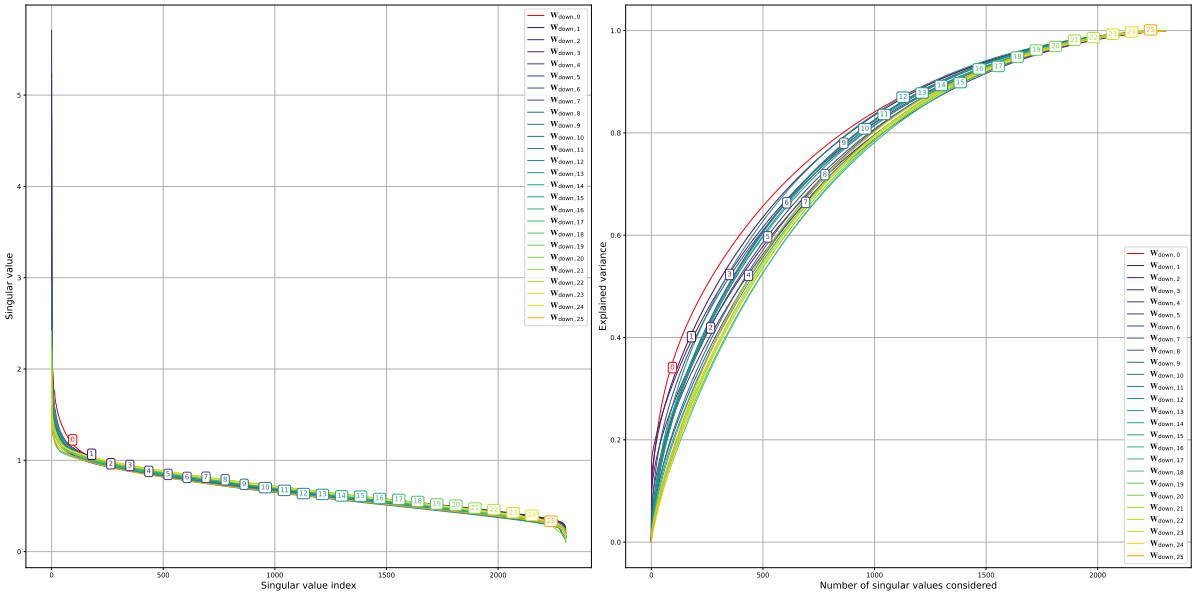


Figure A.15: Singular value distribution and cumulative explained variance of down projection matrices in Gemma 2 (2B) blocks.

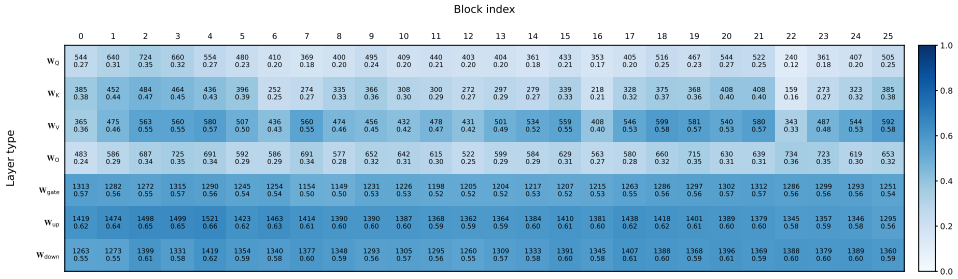


Figure A.16: Approximate ranks of the weight matrices in Gemma 2 (2B), determined using a threshold of 0.9 for the explained variance.

A.1.3. Gemma-2-9b

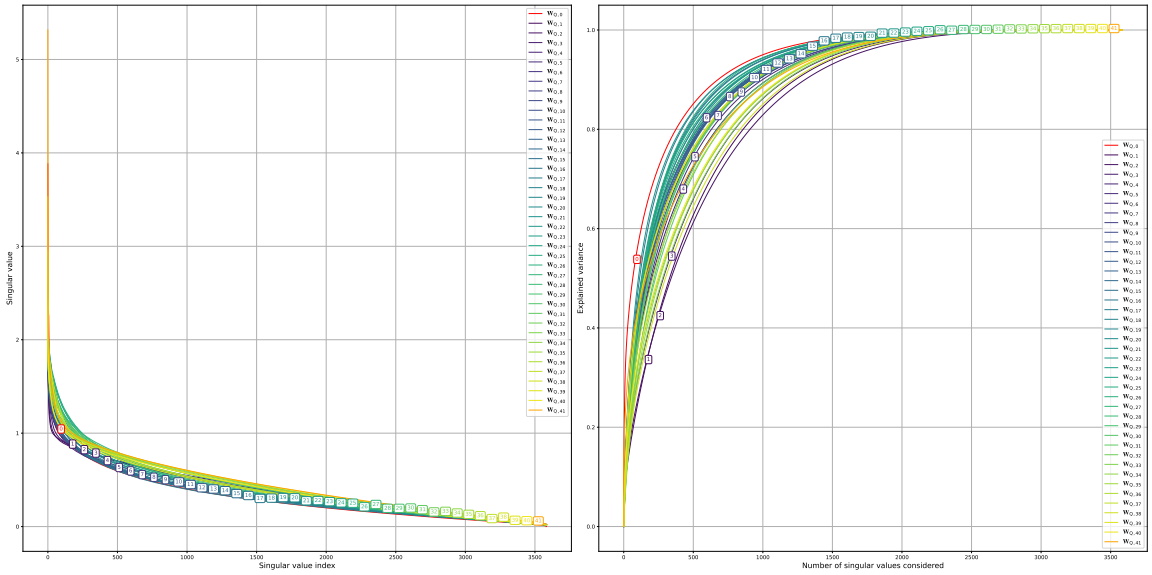


Figure A.17: Singular value distribution and cumulative explained variance of self-attention query matrices in Gemma 2 (9B) blocks.

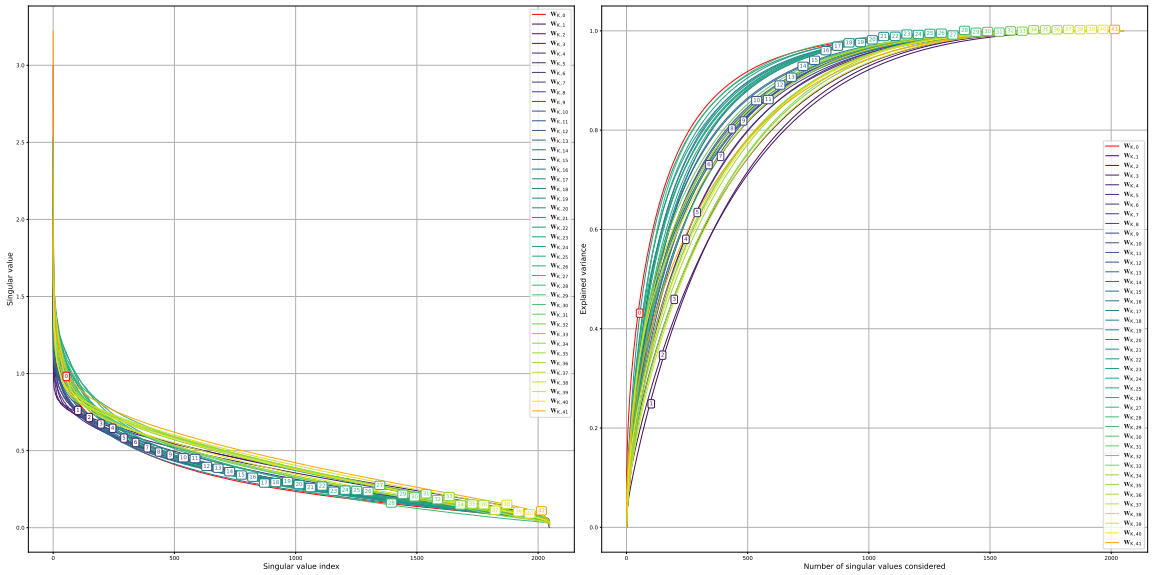


Figure A.18: Singular value distribution and cumulative explained variance of self-attention key matrices in Gemma 2 (9B) blocks.

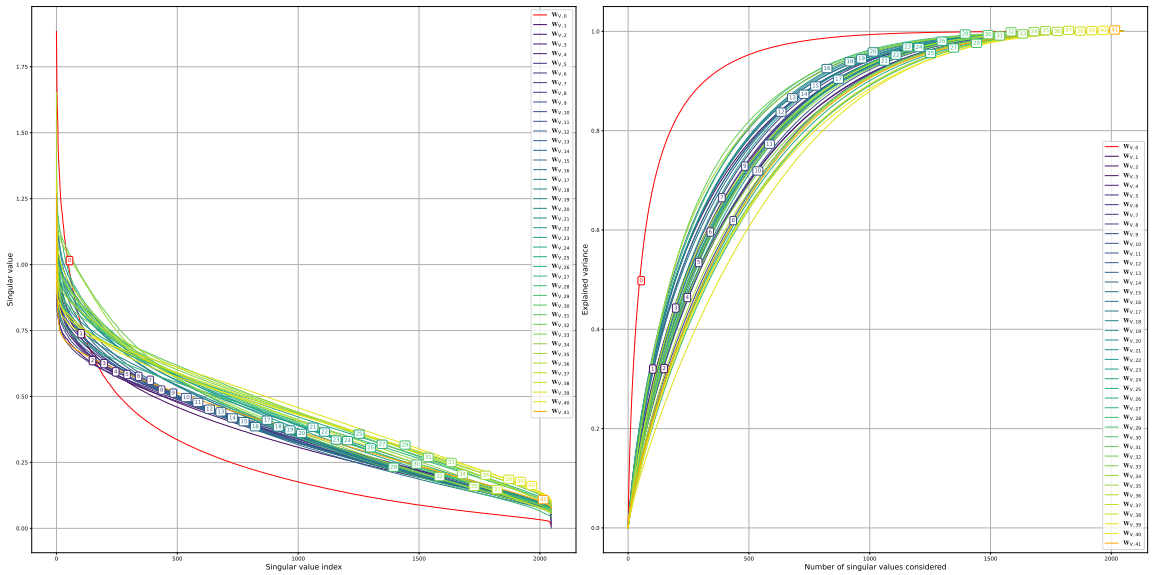


Figure A.19: Singular value distribution and cumulative explained variance of self-attention value matrices in Gemma 2 (9B) blocks.

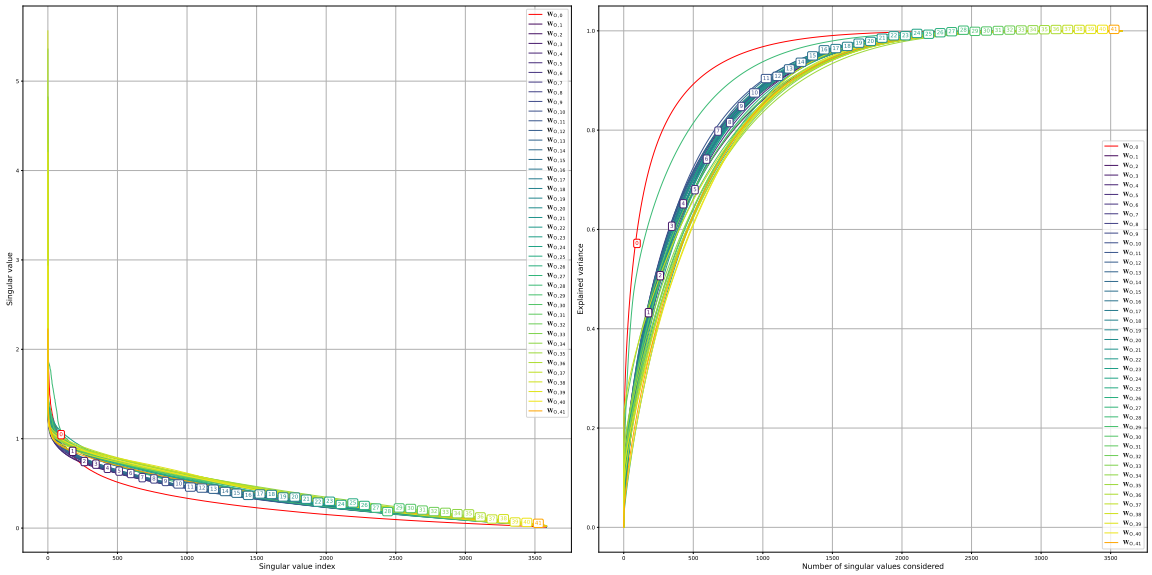


Figure A.20: Singular value distribution and cumulative explained variance of self-attention output matrices in Gemma 2 (9B) blocks.

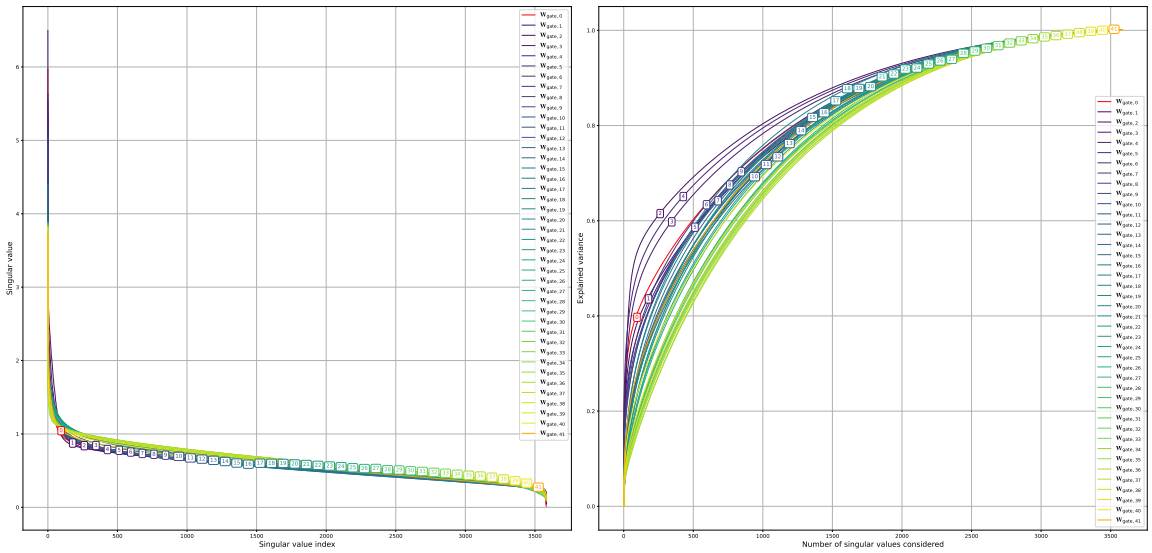


Figure A.21: Singular value distribution and cumulative explained variance of gate projection matrices in Gemma 2 (9B) blocks.

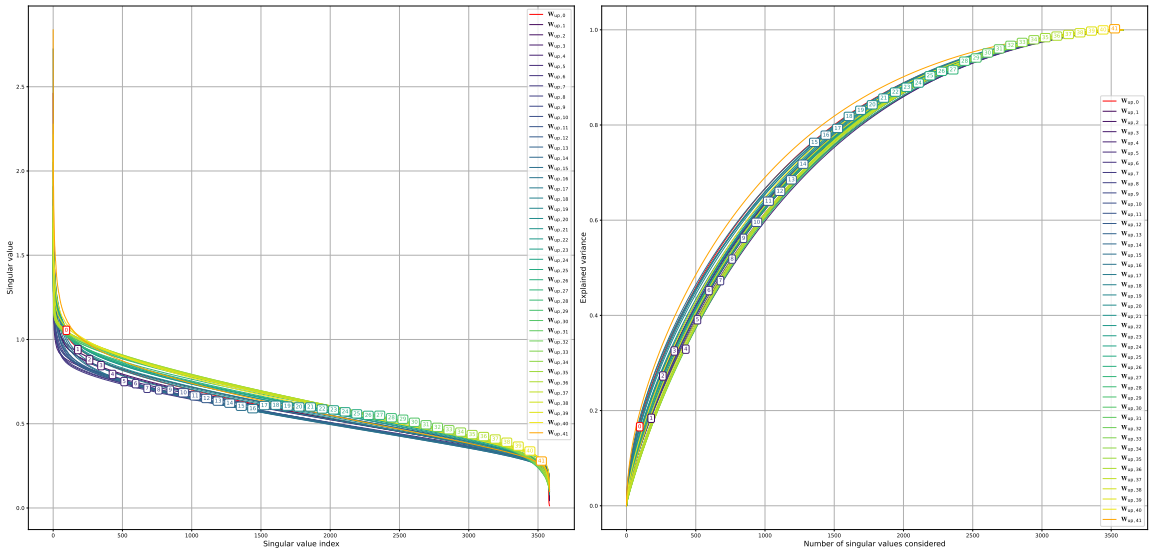


Figure A.22: Singular value distribution and cumulative explained variance of up projection matrices in Gemma 2 (9B) blocks.

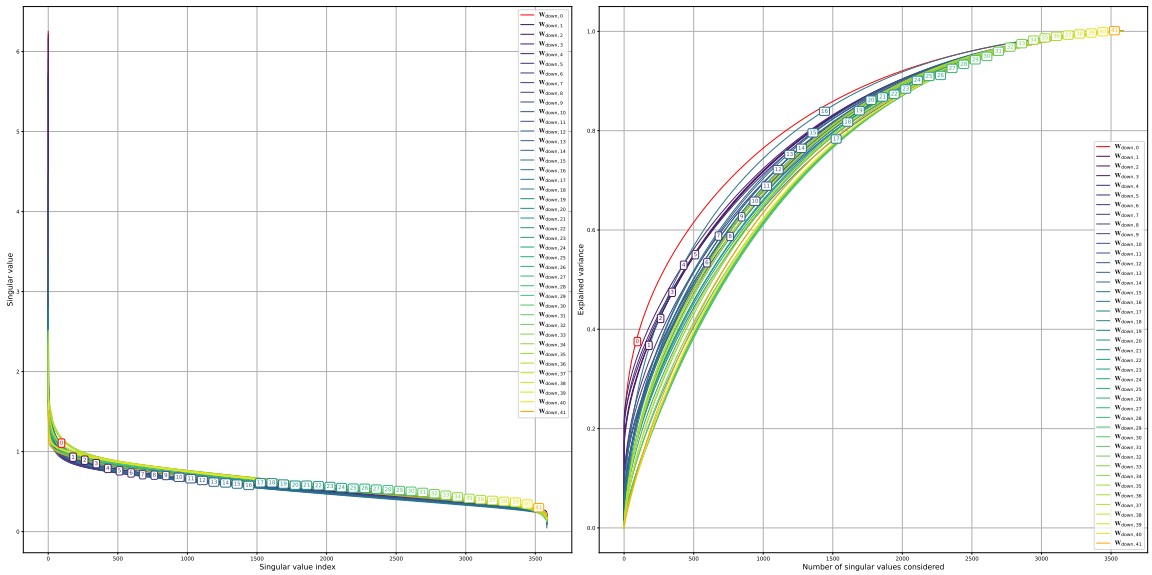


Figure A.23: Singular value distribution and cumulative explained variance of down projection matrices in Gemma 2 (9B) blocks.

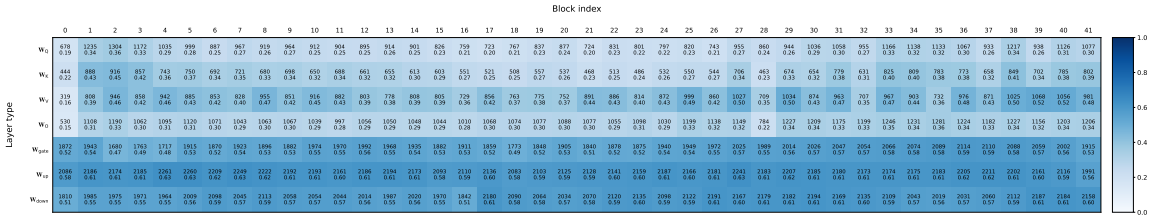


Figure A.24: Approximate ranks of the weight matrices in gemma-2-9b.

A.2. Inter-Layers Rank Analysis

This section explores redundancy across multiple layers within the same model. The methodology, outlined in Section 7.1.2, involves concatenating matrices of the same type from different blocks and computing their singular values and approximate ranks.

Due to computational constraints, the analysis of concatenated matrices was performed only for Mistral-7B-v0.3 and Gemma-2-2B. The larger size of the Gemma-2-9B matrices made it impractical to perform the same computations with the available resources.

A.2.1. Mistral-7B-v0.3

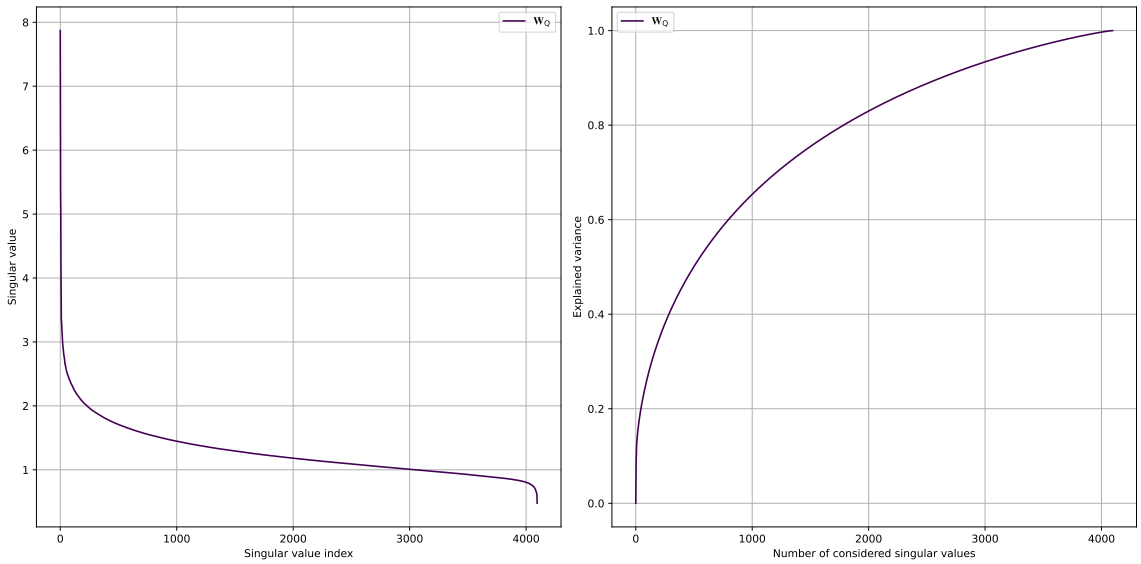


Figure A.25: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention query matrices in Mistral blocks.

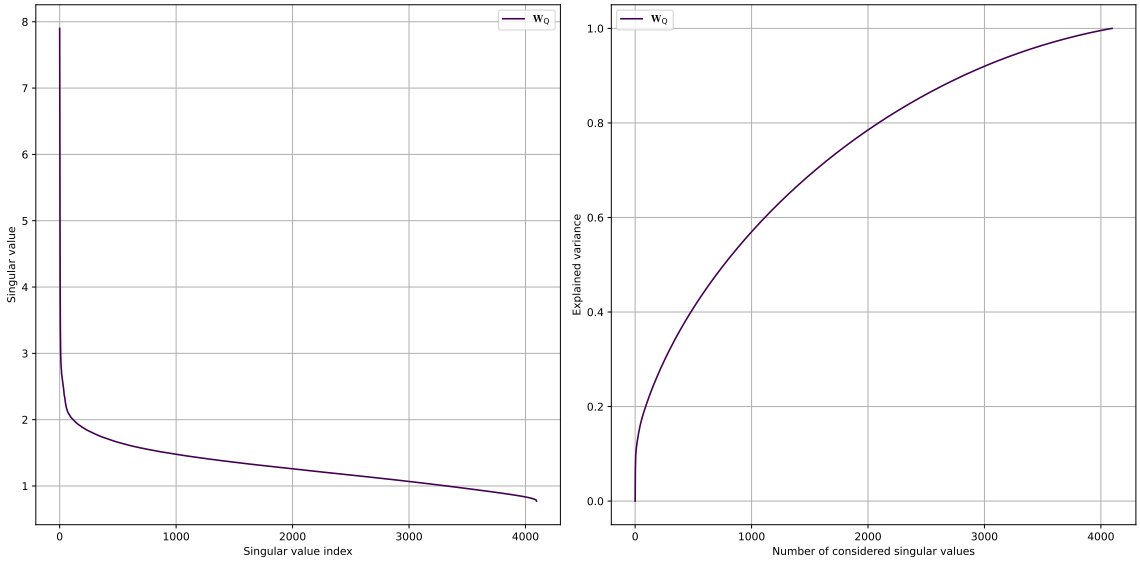


Figure A.26: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention query matrices in Mistral blocks.

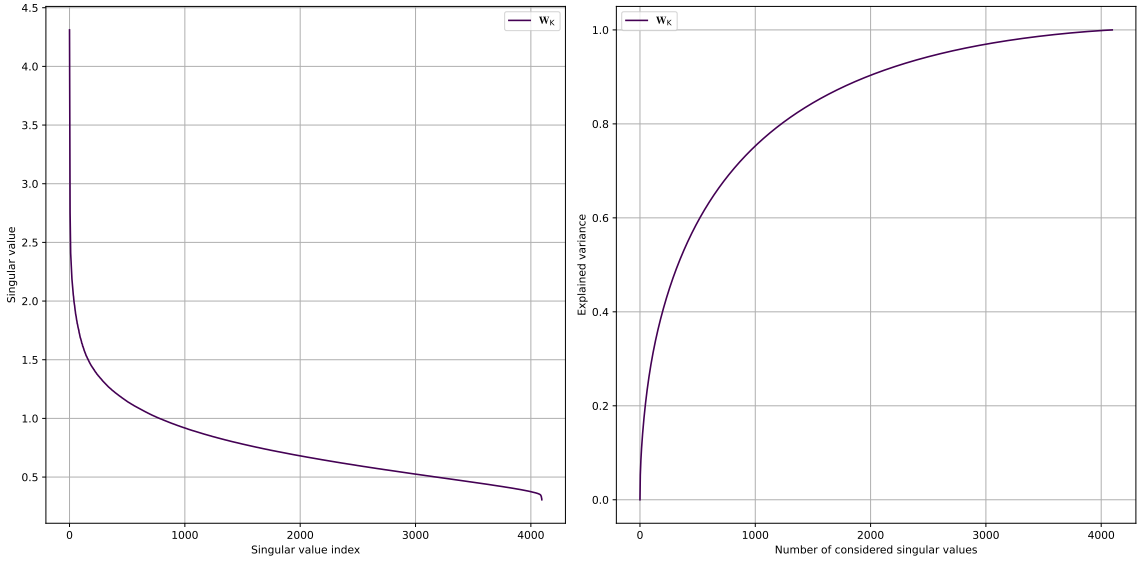


Figure A.27: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention key matrices in Mistral blocks.

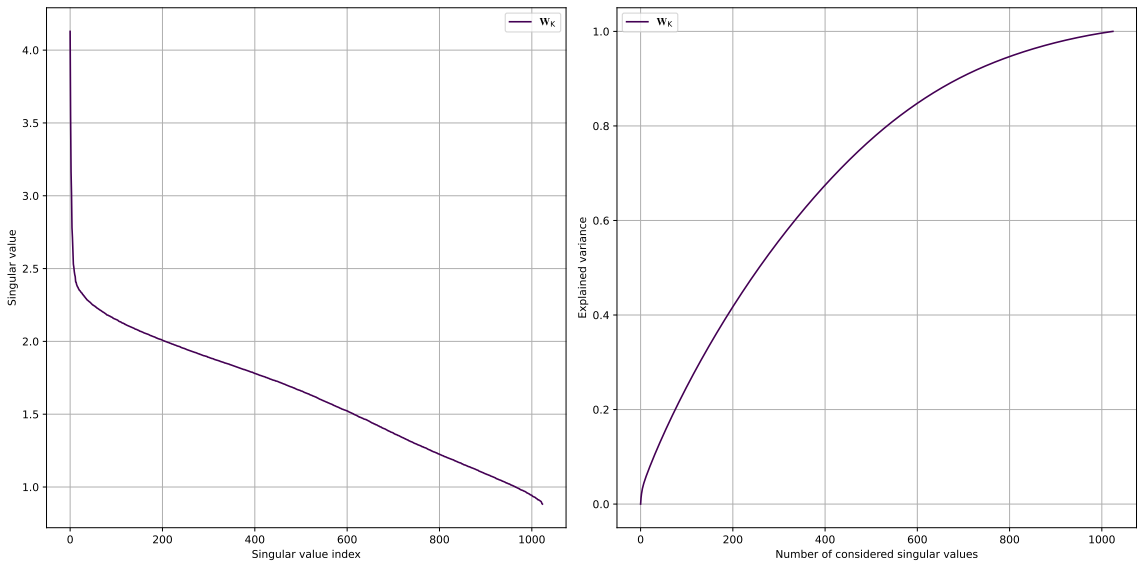


Figure A.28: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention key matrices in Mistral blocks.

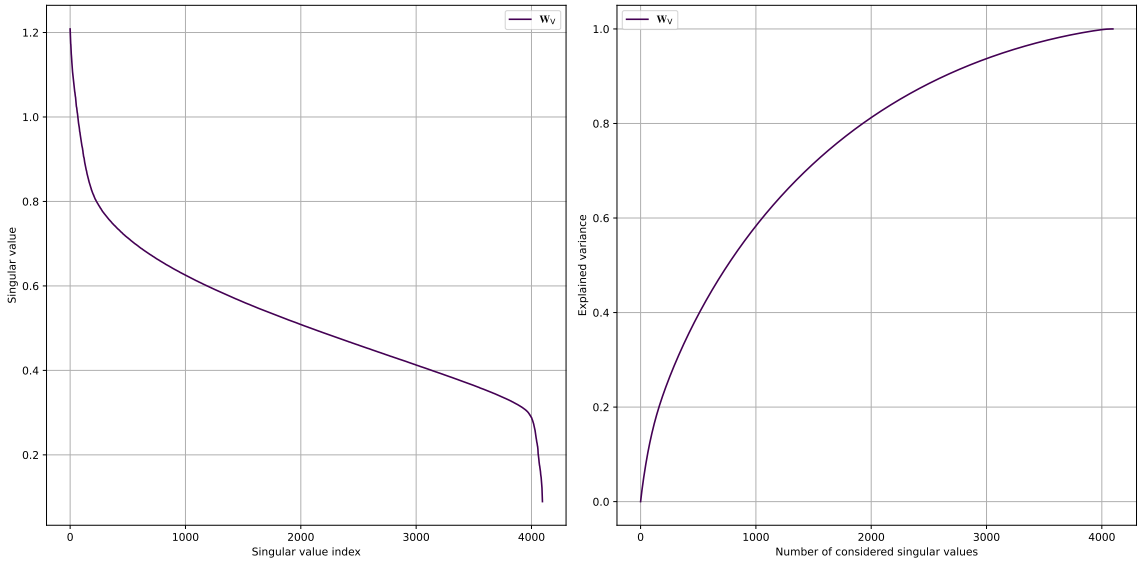


Figure A.29: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention value matrices in Mistral blocks.

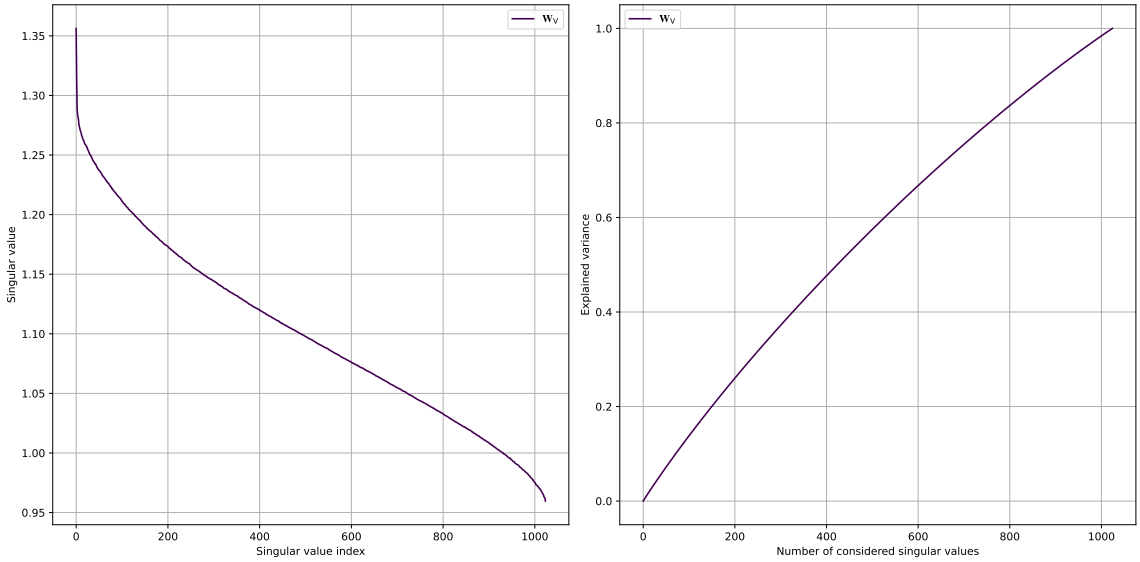


Figure A.30: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention value matrices in Mistral blocks.

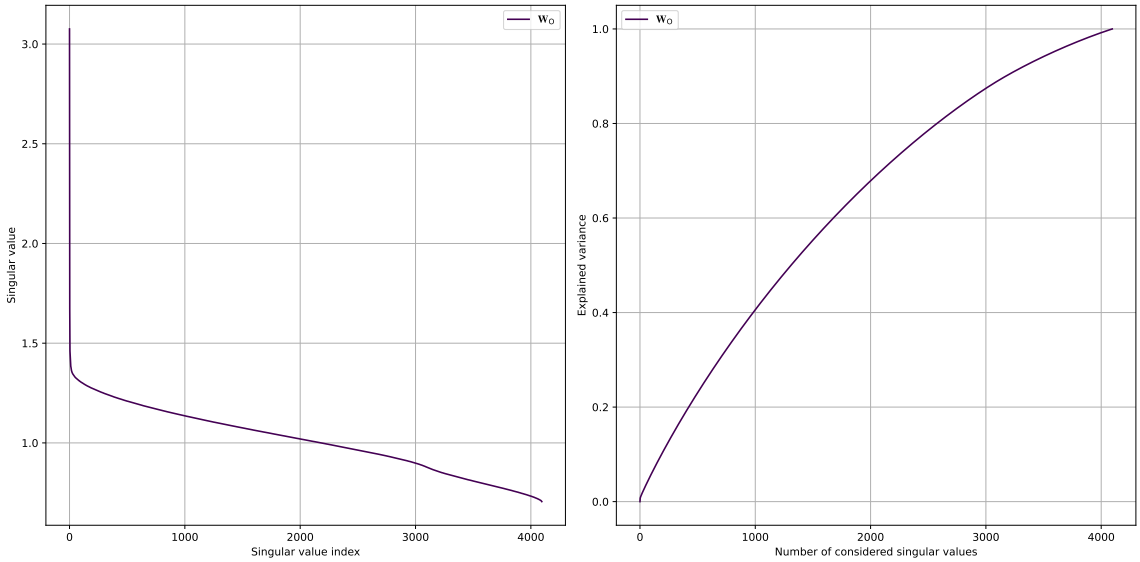


Figure A.31: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention output matrices in Mistral blocks.

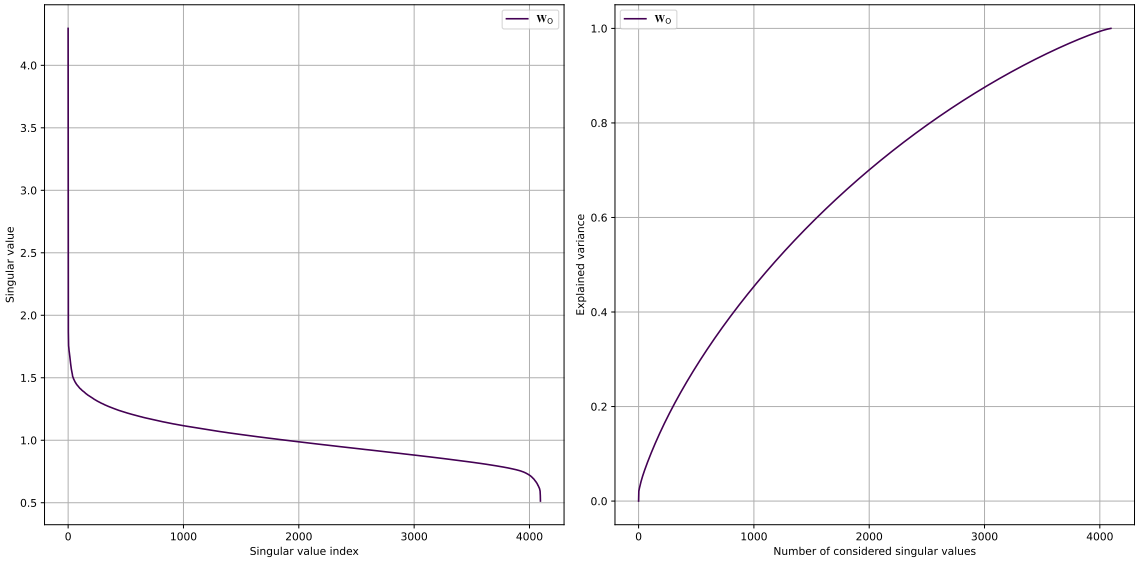


Figure A.32: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention output matrices in Mistral blocks.

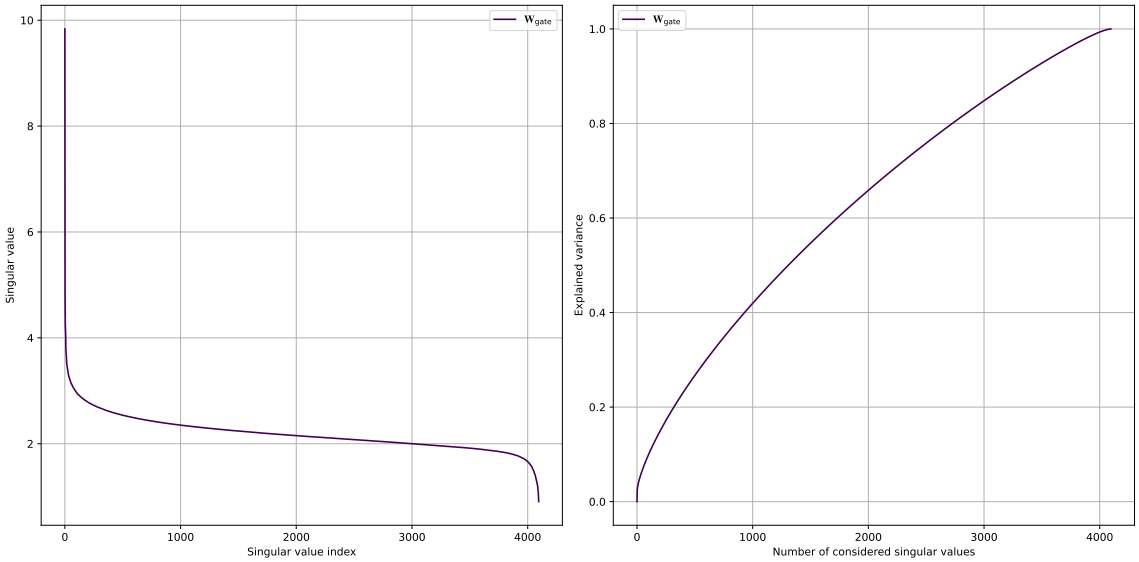


Figure A.33: Singular value distribution and cumulative explained variance of the row-wise concatenation of gate projection matrices in Mistral blocks.

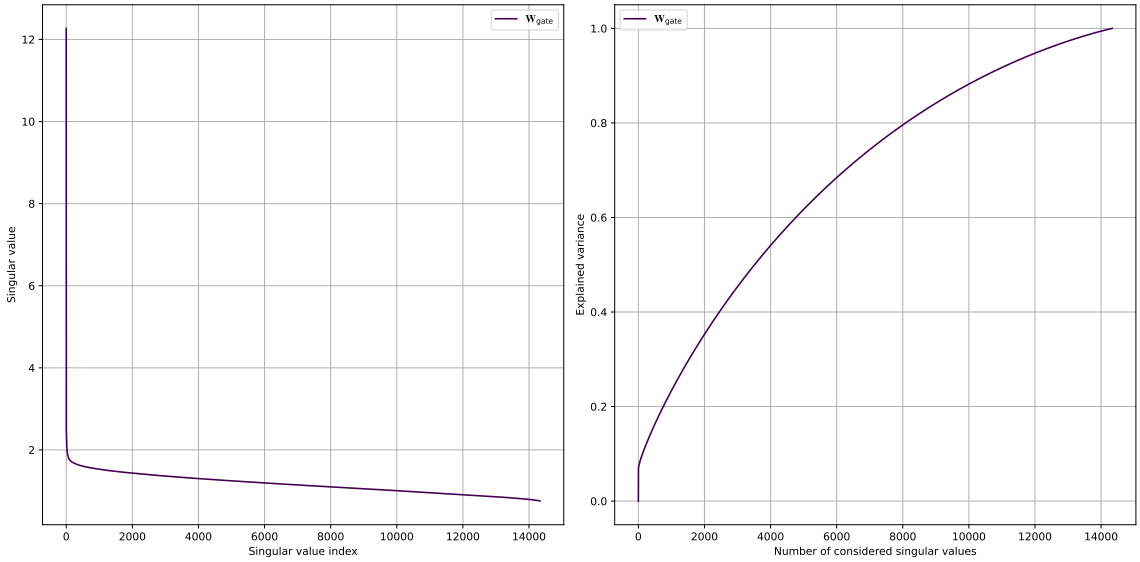


Figure A.34: Singular value distribution and cumulative explained variance of the column-wise concatenation of gate projection matrices in Mistral blocks.

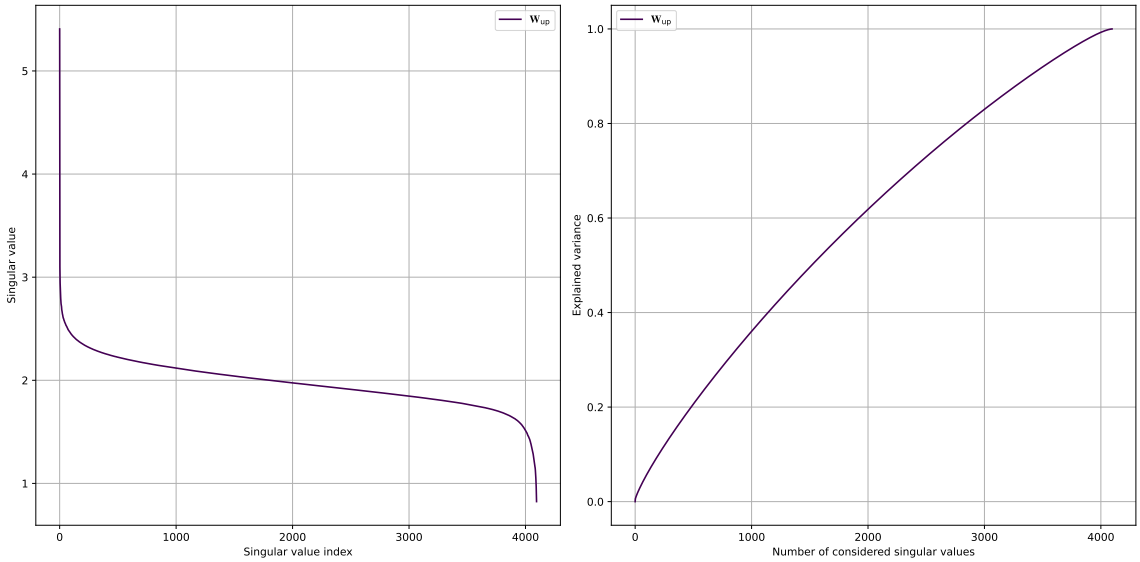


Figure A.35: Singular value distribution and cumulative explained variance of the row-wise concatenation of up projection matrices in Mistral blocks.

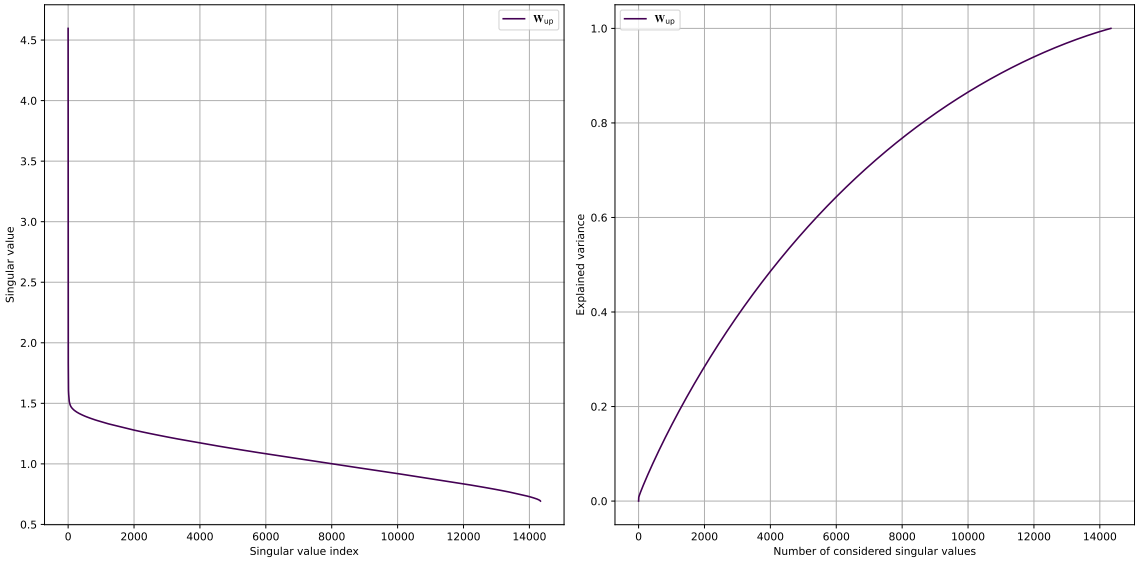


Figure A.36: Singular value distribution and cumulative explained variance of the column-wise concatenation of up projection matrices in Mistral blocks.

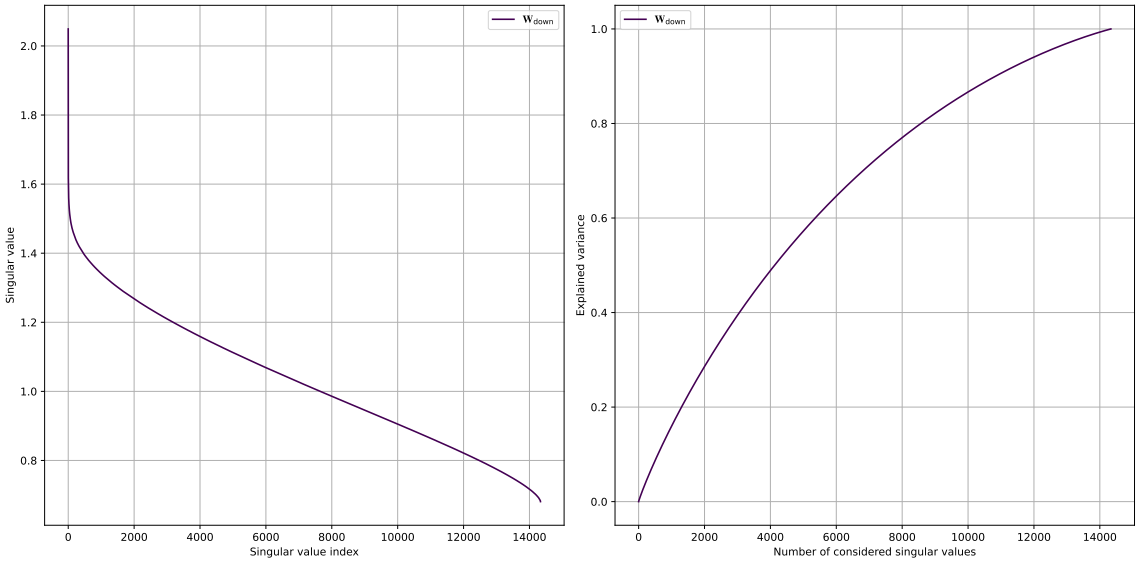


Figure A.37: Singular value distribution and cumulative explained variance of the row-wise concatenation of down projection matrices in Mistral blocks.

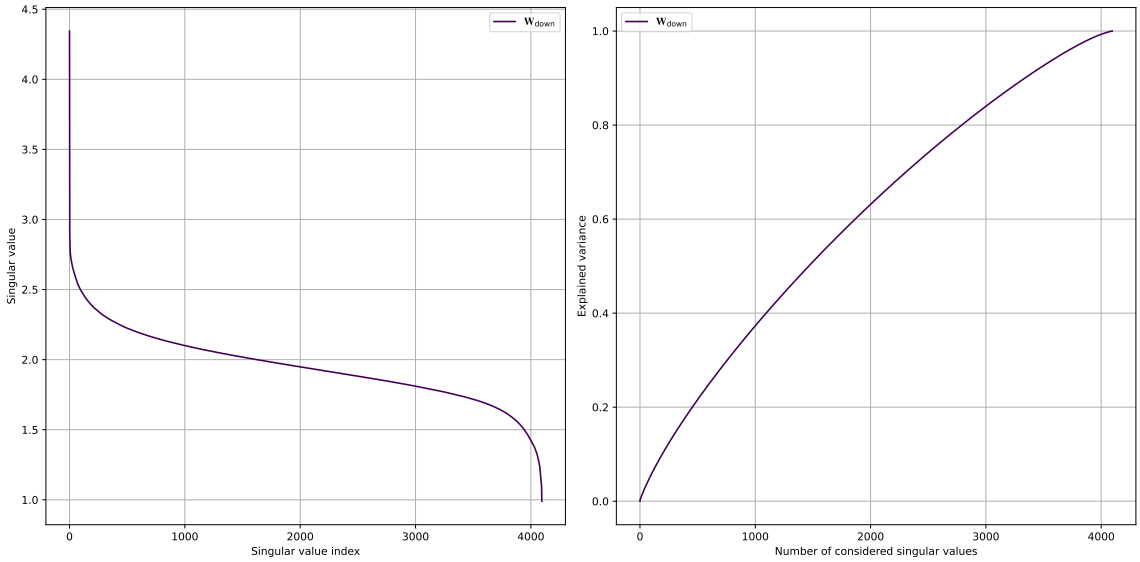


Figure A.38: Singular value distribution and cumulative explained variance of the column-wise concatenation of down projection matrices in Mistral blocks.

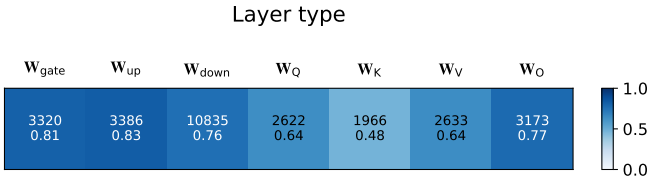


Figure A.39: Approximate ranks of the weight matrices in Mistral row-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

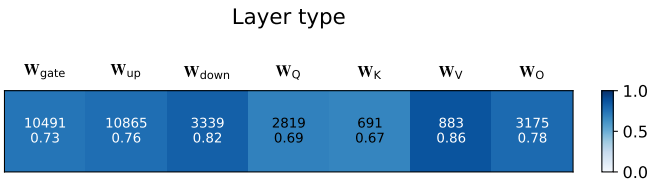


Figure A.40: Approximate ranks of the weight matrices in Mistral column-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

A.2.2. Gemma-2-2b

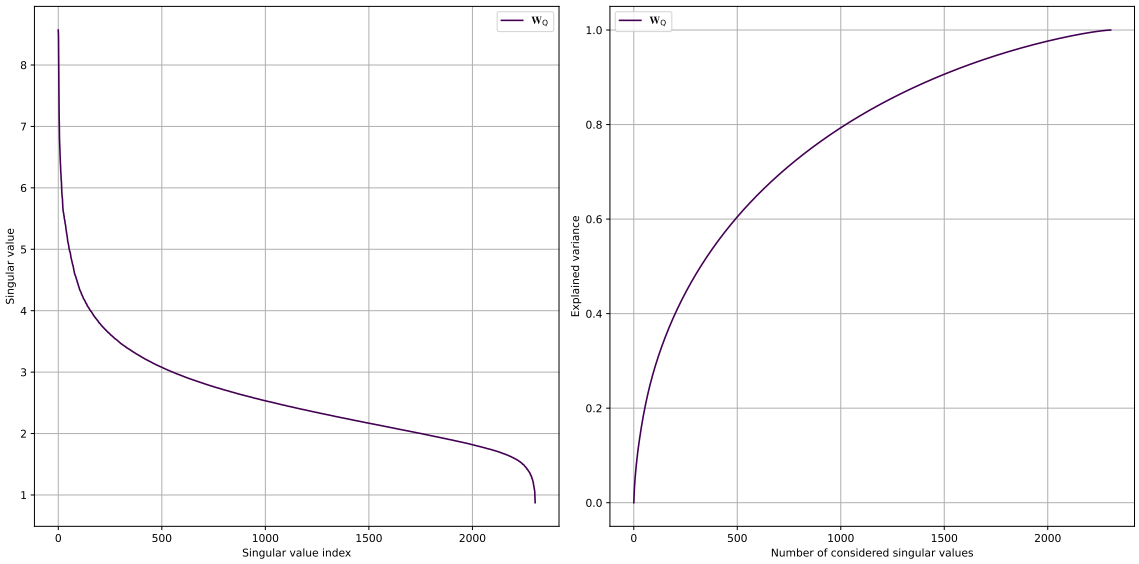


Figure A.41: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention query matrices in Gemma 2 (2B) blocks.

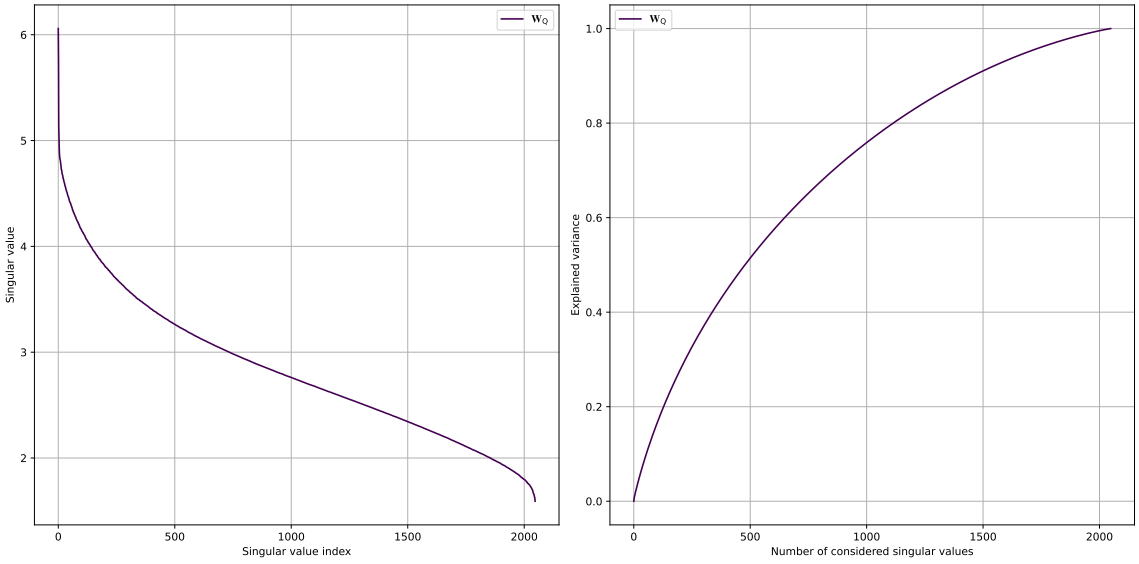


Figure A.42: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention query matrices in Gemma 2 (2B) blocks.

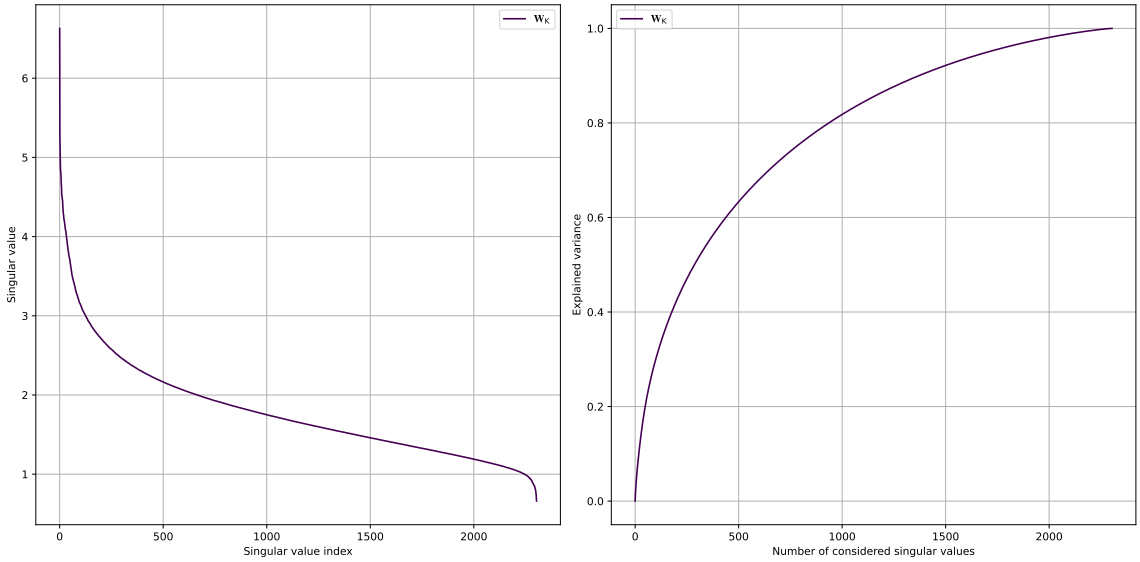


Figure A.43: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention key matrices in Gemma 2 (2B) blocks.

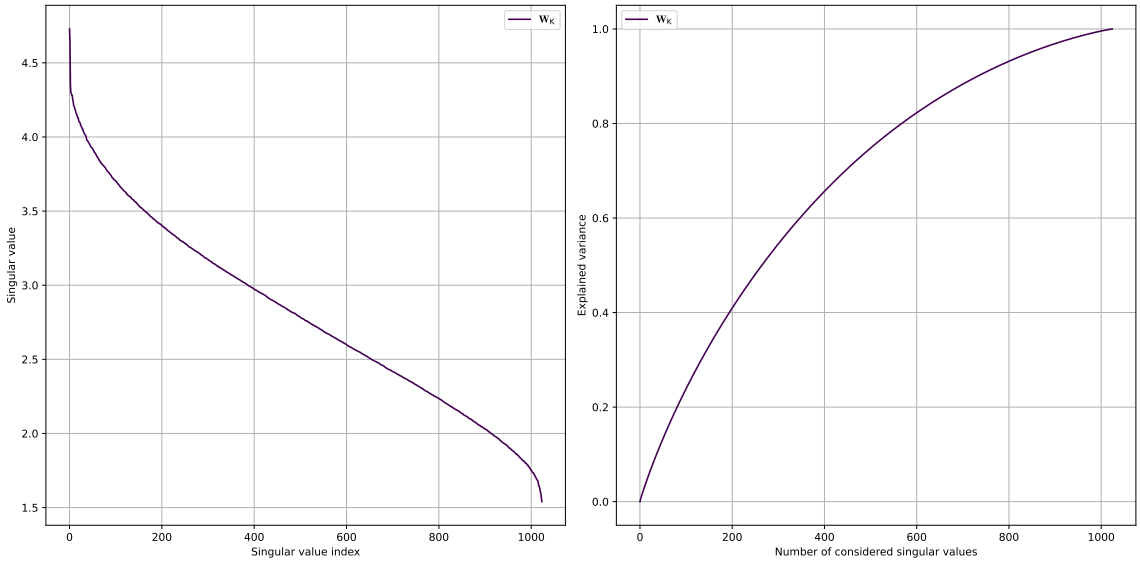


Figure A.44: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention key matrices in Gemma 2 (2B) blocks.

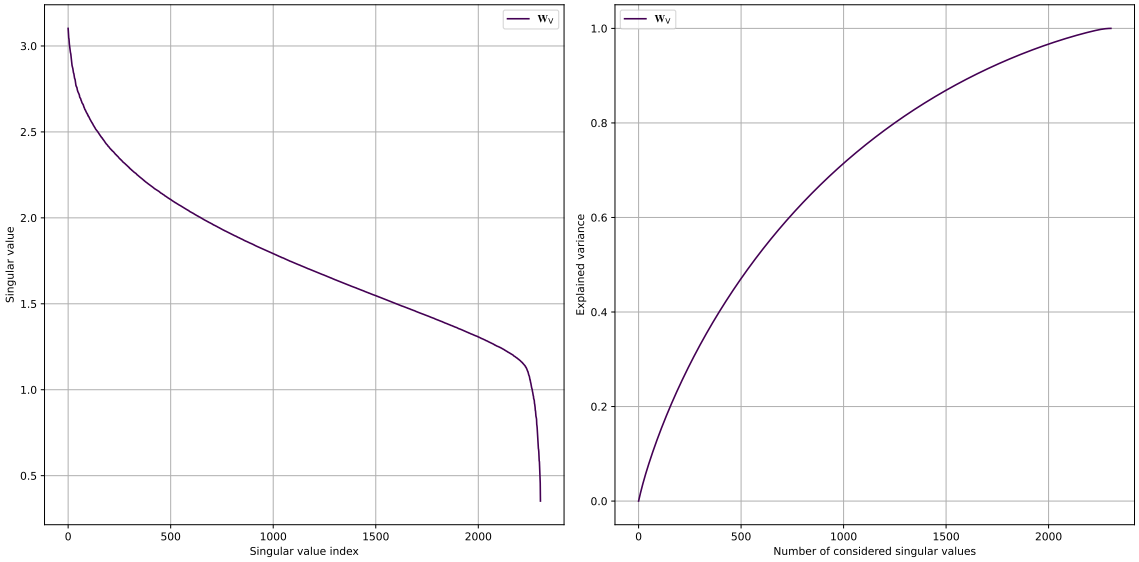


Figure A.45: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention value matrices in Gemma 2 (2B) blocks.

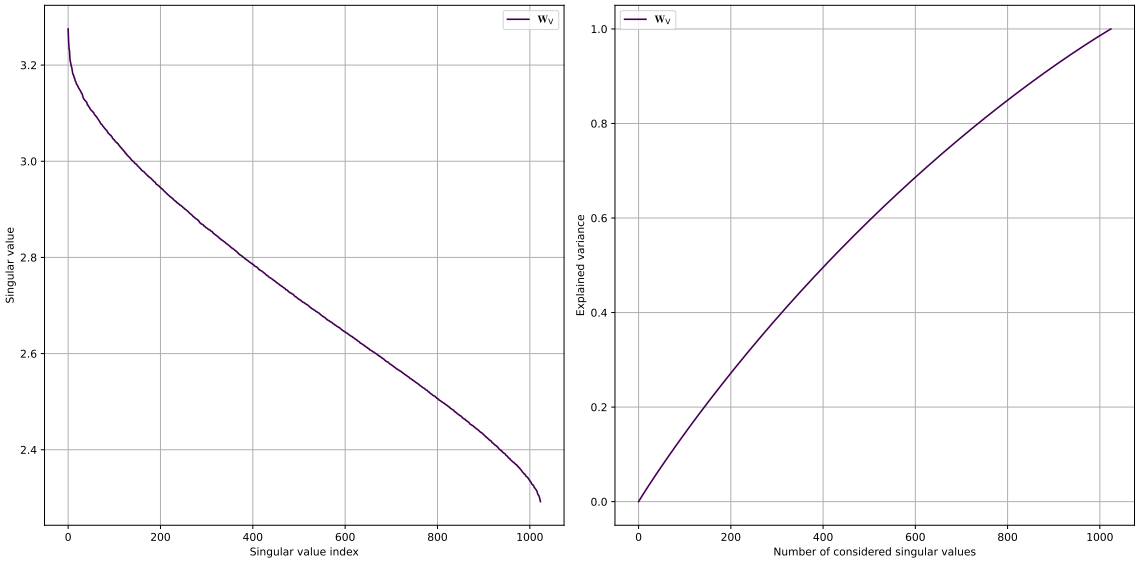


Figure A.46: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention value matrices in Gemma 2 (2B) blocks.

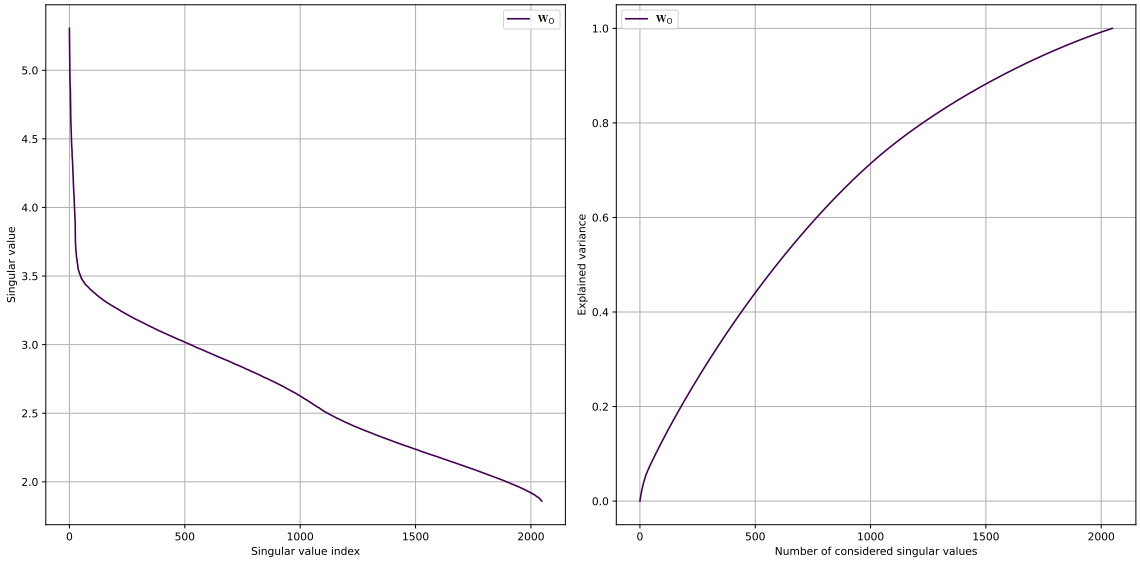


Figure A.47: Singular value distribution and cumulative explained variance of the row-wise concatenation of self-attention output matrices in Gemma 2 (2B) blocks.

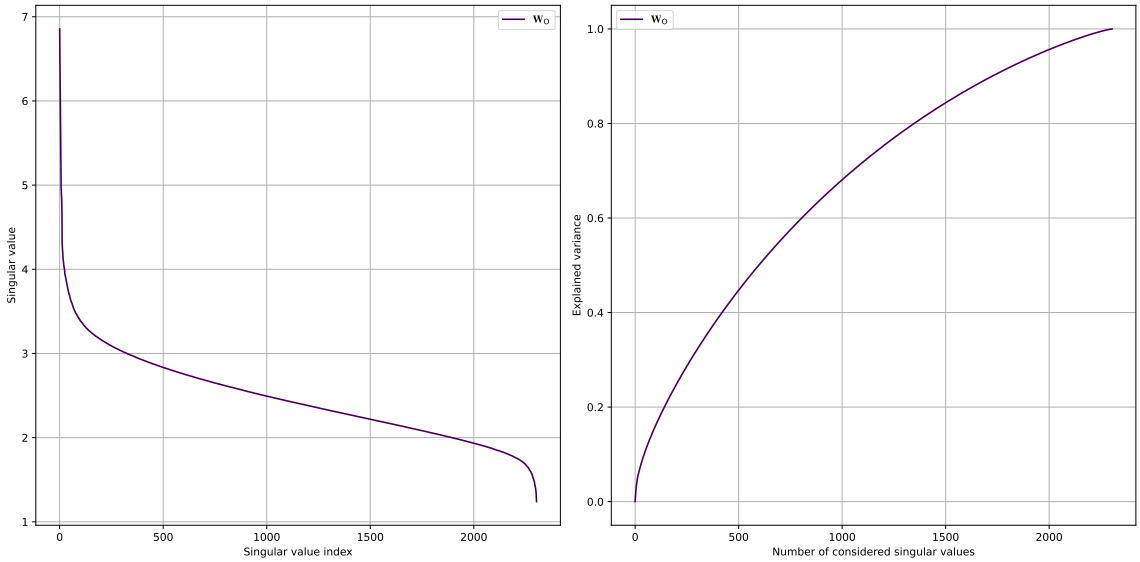


Figure A.48: Singular value distribution and cumulative explained variance of the column-wise concatenation of self-attention output matrices in Gemma 2 (2B) blocks.

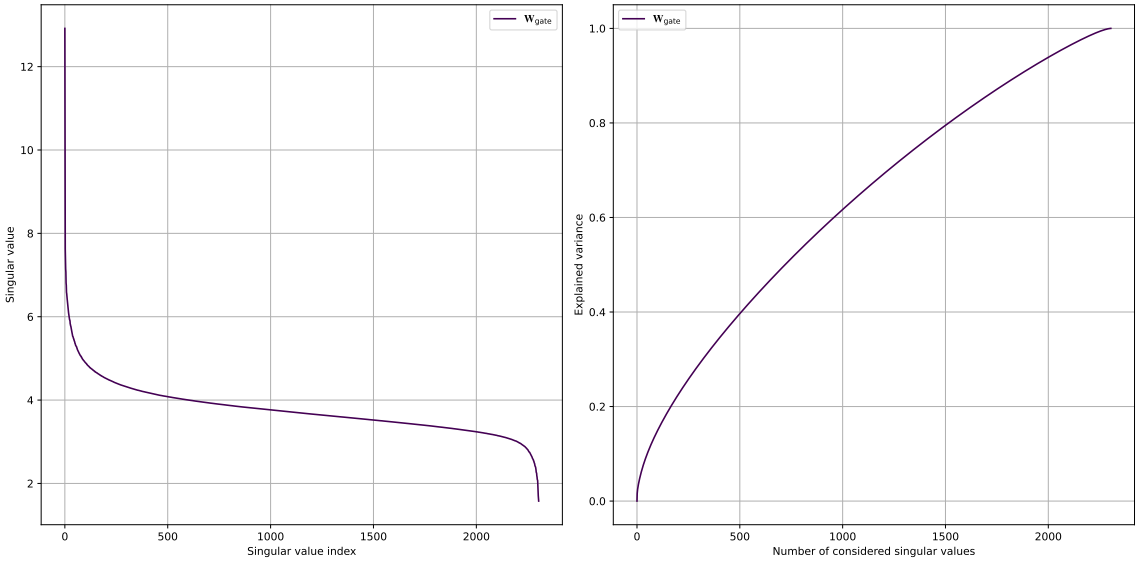


Figure A.49: Singular value distribution and cumulative explained variance of the row-wise concatenation of gate projection matrices in Gemma 2 (2B) blocks.

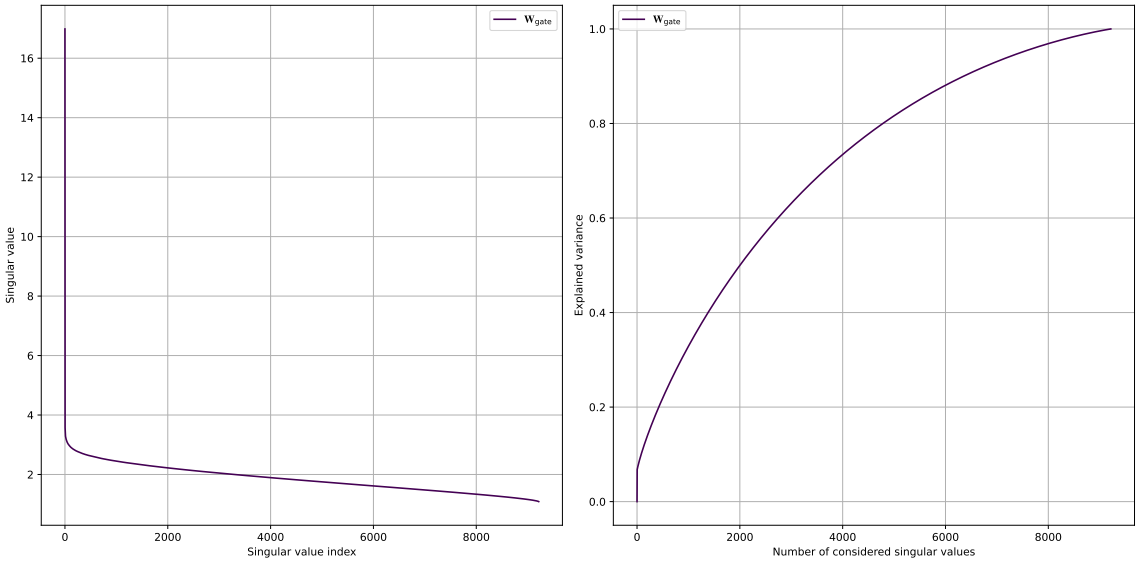


Figure A.50: Singular value distribution and cumulative explained variance of the column-wise concatenation of gate projection matrices in Gemma 2 (2B) blocks.

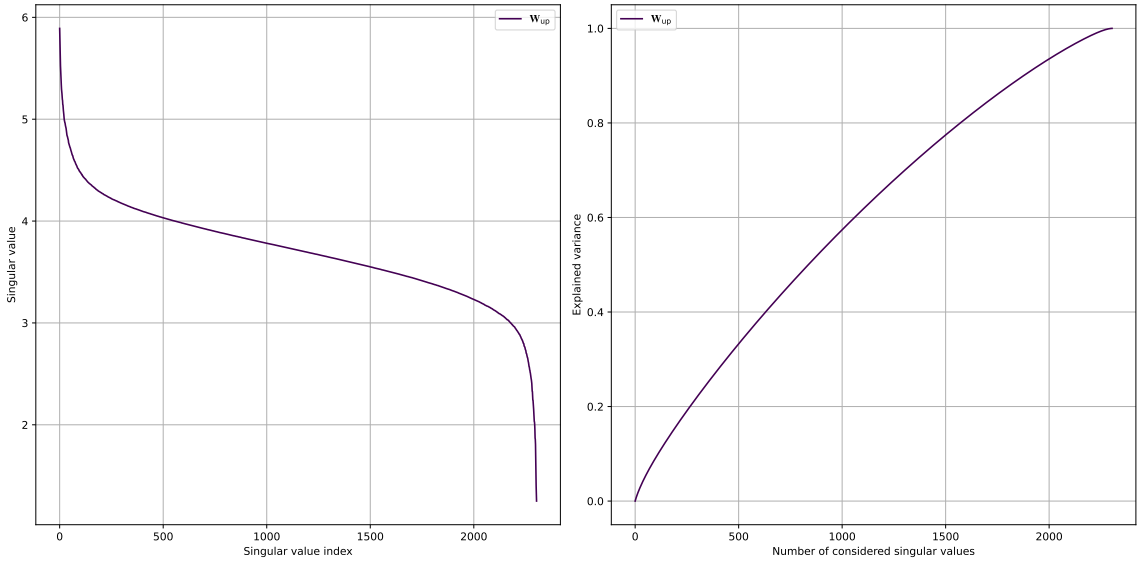


Figure A.51: Singular value distribution and cumulative explained variance of the row-wise concatenation of up projection matrices in Gemma 2 (2B) blocks.

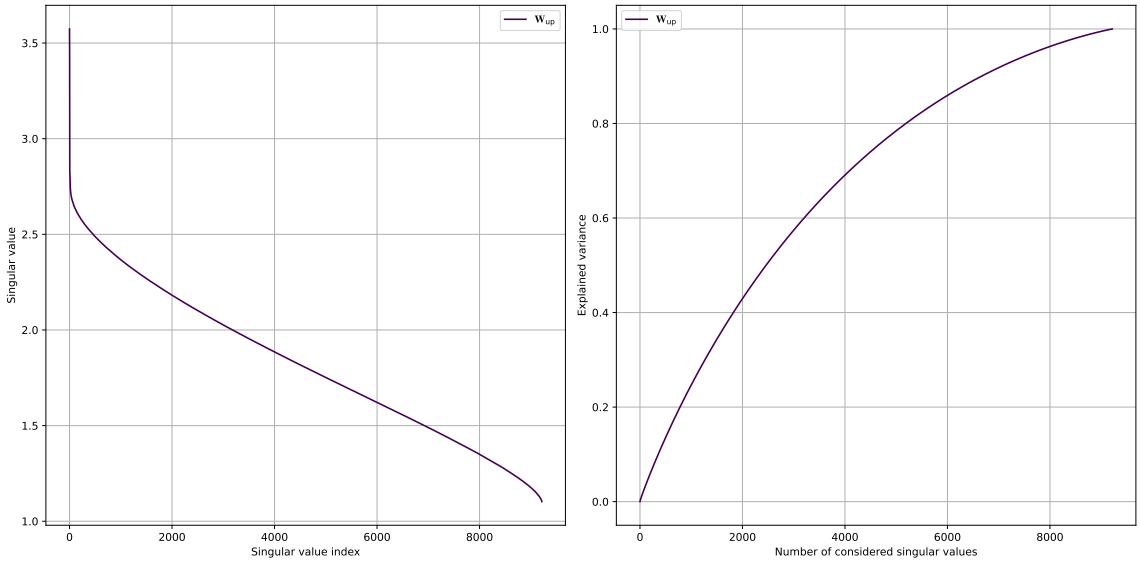


Figure A.52: Singular value distribution and cumulative explained variance of the column-wise concatenation of up projection matrices in Gemma 2 (2B) blocks.

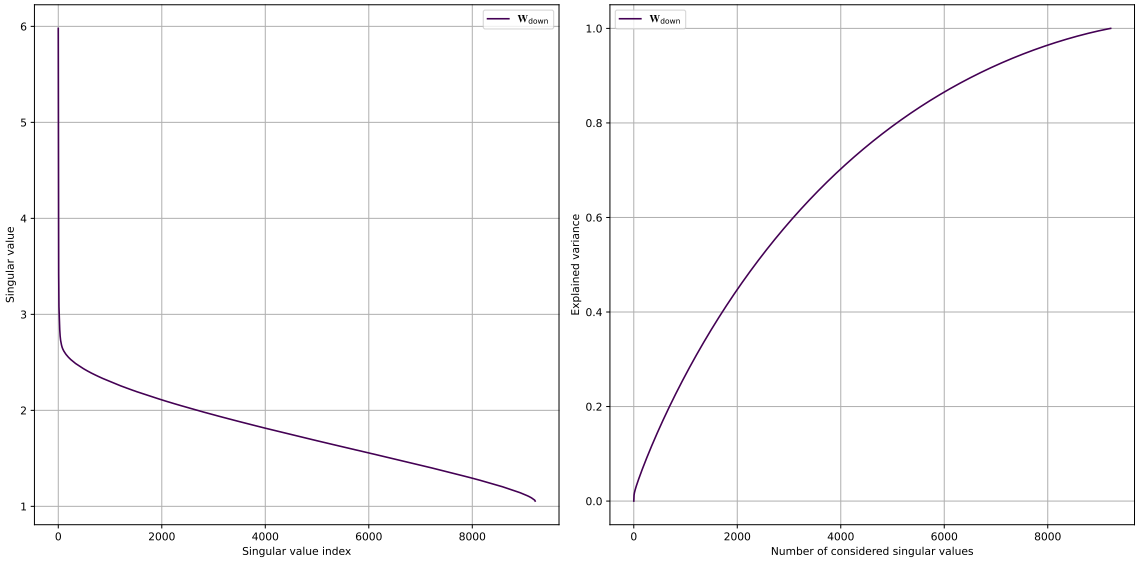


Figure A.53: Singular value distribution and cumulative explained variance of the row-wise concatenation of down projection matrices in Gemma 2 (2B) blocks.

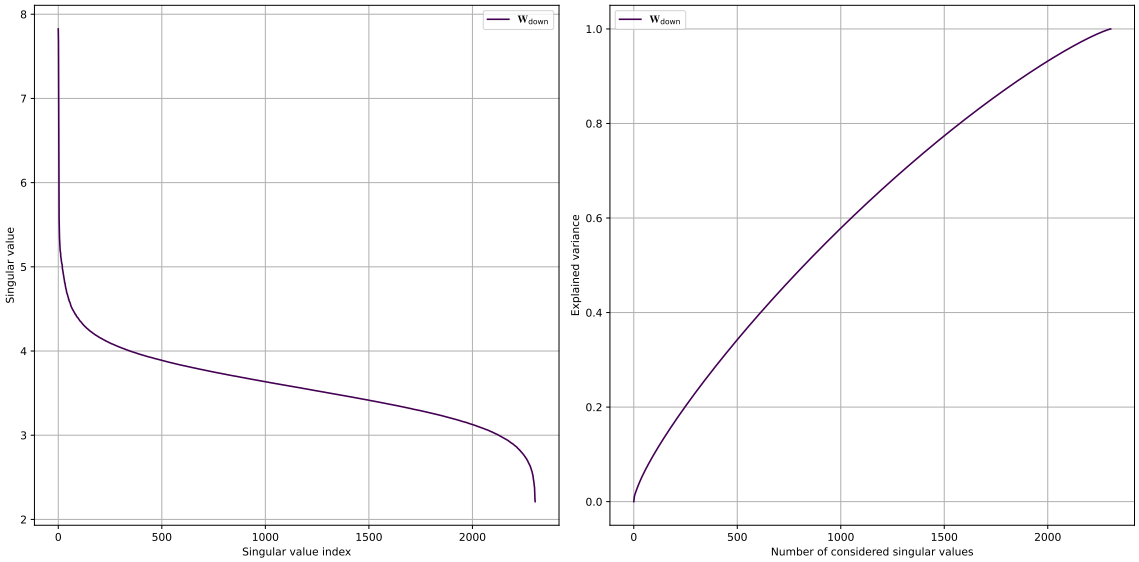


Figure A.54: Singular value distribution and cumulative explained variance of the column-wise concatenation of down projection matrices in Gemma 2 (2B) blocks.

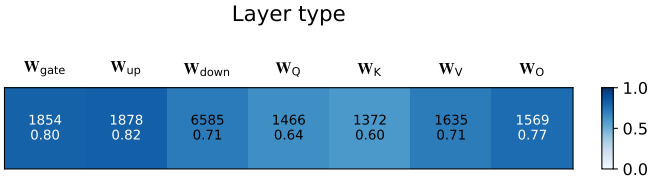


Figure A.55: Approximate ranks of the weight matrices in Mistral row-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

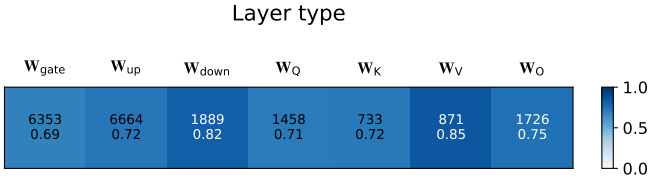


Figure A.56: Approximate ranks of the weight matrices in Mistral column-wise concatenated based on their role, determined using a threshold of 0.9 for the explained variance.

B | Appendix B

Appendix B expands on Section 7.1.3, presenting additional results from the module replacement analysis applied to Transformer components. It extends the investigation beyond Llama 3.1 to other architectures. Due to the high computational cost of benchmarking each modified model, only partial results are reported for the self-attention components of LLMs.

B.1. Llama-3.1-8B

This section presents the results of the layer replacement analysis applied to the self-attention modules of Llama 3.1 (8B). Following the methodology outlined in Section 7.1.3, it offers insights into the functional redundancy of these modules.

Characterization of the redundancy in self-attention modules

Llama 3.1 with self-attention layers replaced by those of another block and the one with ablated layers perform similarly and remain close to the performance of the original model on HellaSwag (Figure B.3). In the central blocks, replacement slightly outperforms ablation. A notable exception occurs in the initial blocks, where replacing self-attention leads to significantly worse performance than ablation. This shows that the self-attention modules in the first blocks of Llama 3.1 perform highly specialized and critical functions that are not replicated by other layers.

A similar trend for the first self-attention modules is observed in Figure B.6. In the case of performance deltas on GSM8K, a higher concentration of positive values appears in the heatmap, particularly in columns corresponding to high values on the main diagonal. In many instances, with the exception of the first layers once again, substituting the self-attention mechanisms proves to be less disruptive than removing the self-attention mechanism entirely.

When addressing Research Questions 1 and 3, discussed in Sections 4.1 and 4.3, the findings indicate that self-attention mechanisms from different blocks exhibit a partial

degree of similarity and shareability across blocks. However, given that the replacement of the module provides only a slight improvement over a weak baseline where no operation is performed by the self-attention and that the analysis does not offer a comprehensive assessment of all the possible replacements, these conclusions should be interpreted with caution.

Therefore, due to the limited scope of the experiments, weak overall performance, and the lack of robust patterns, it remains difficult to establish a clear rule for layer similarity. Moreover, the answer to Research Question 5 regarding whether layers closer in the stack exhibit greater similarity remains inconclusive.

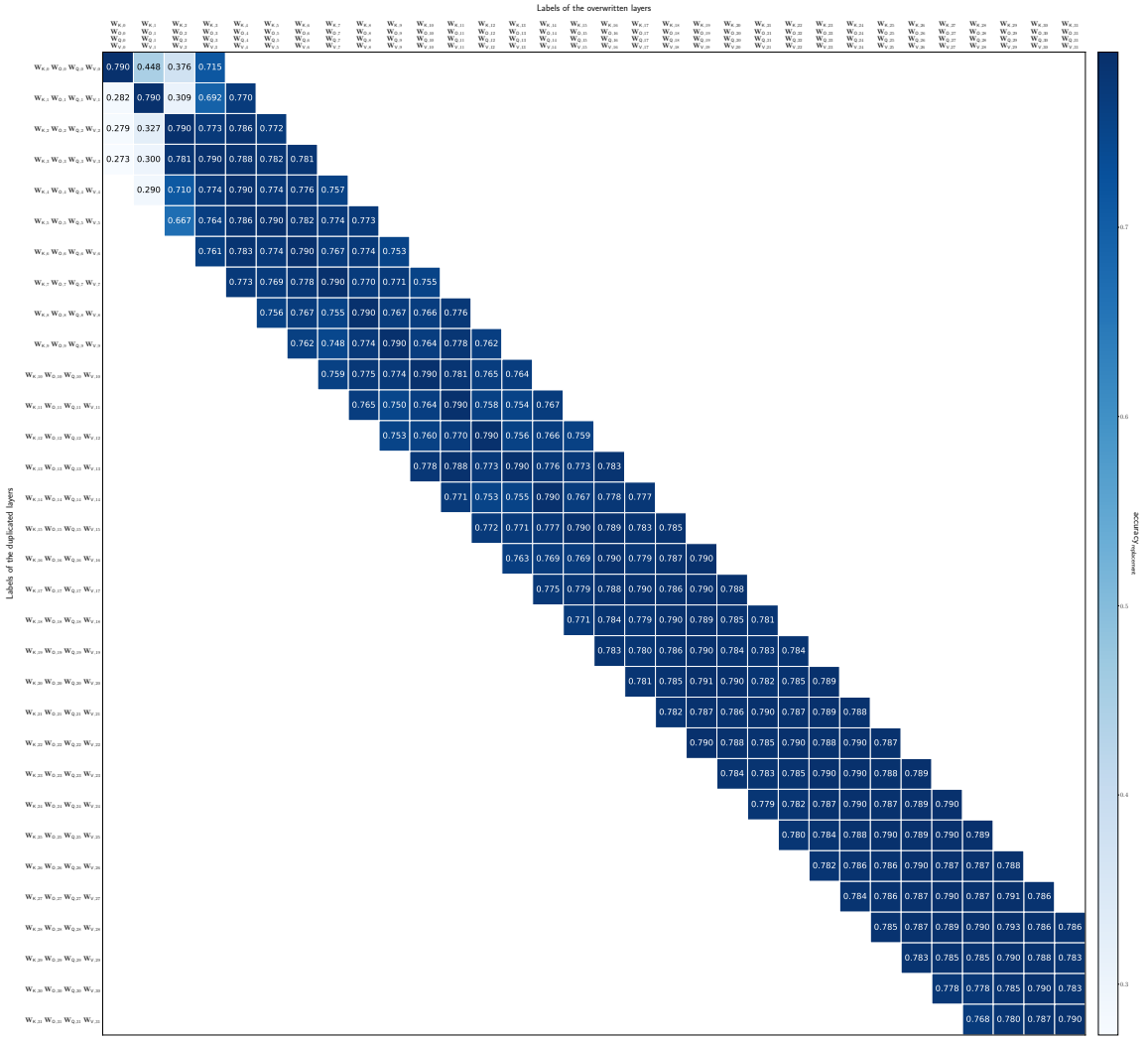


Figure B.1: Heatmap illustrating the accuracy on HellaSwag of Llama 3.1 having linear layers of the self-attention module replaced by the ones of a different block.

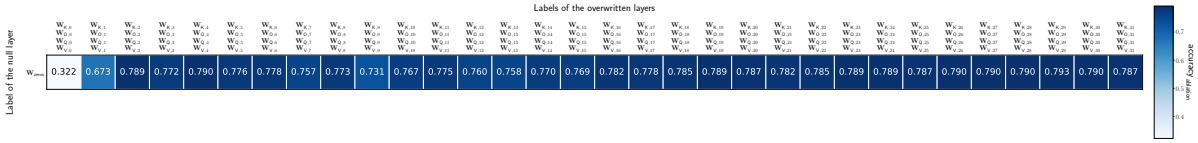


Figure B.2: Heatmap illustrating the accuracy on HellaSwag of Llama 3.1 having linear layers of the self-attention module removed.

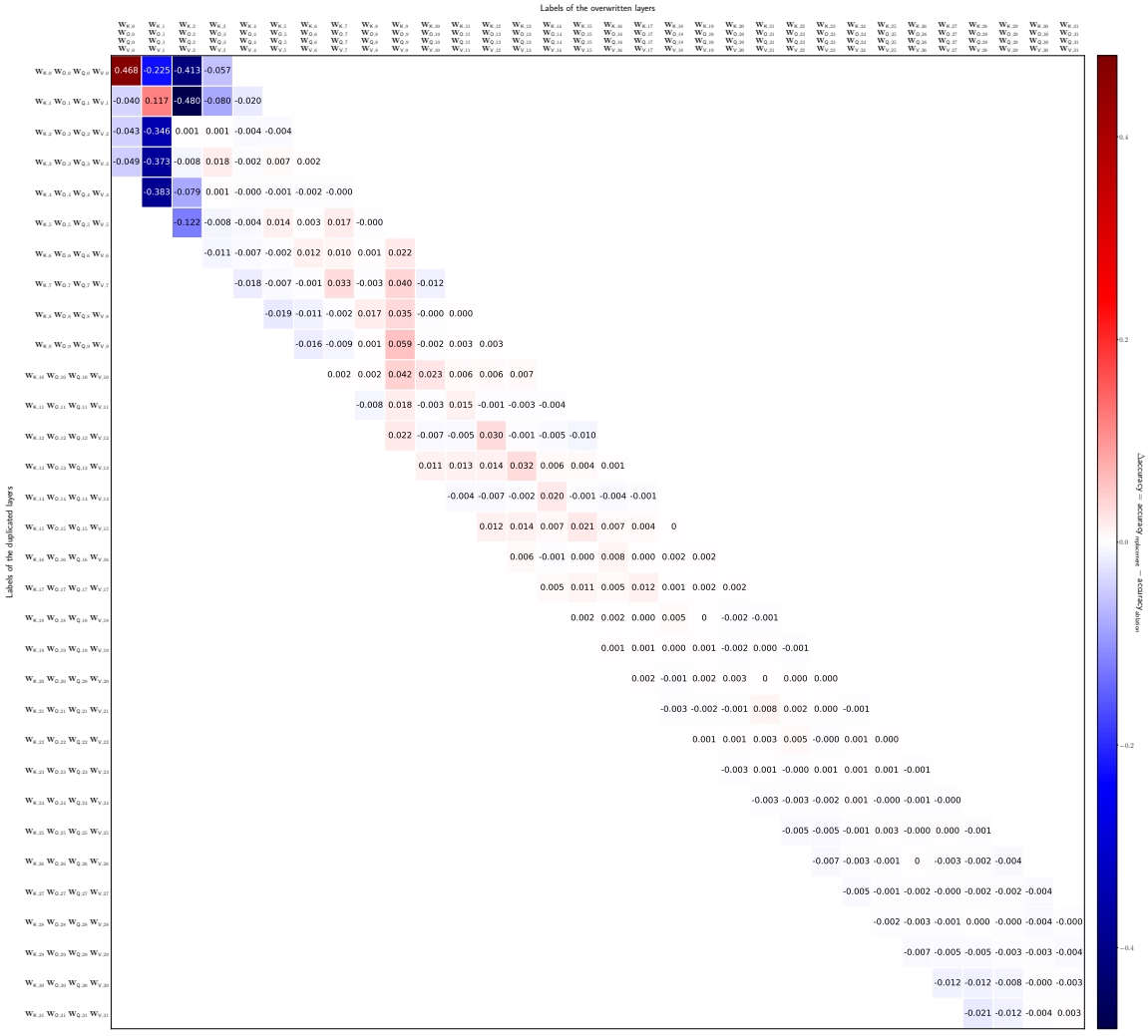


Figure B.3: Heatmap illustrating the difference in performance evaluated on HellaSwag between Llama 3.1 with linear layers in the self-attention module replaced by those from another block and Llama 3.1 with the same layers removed.

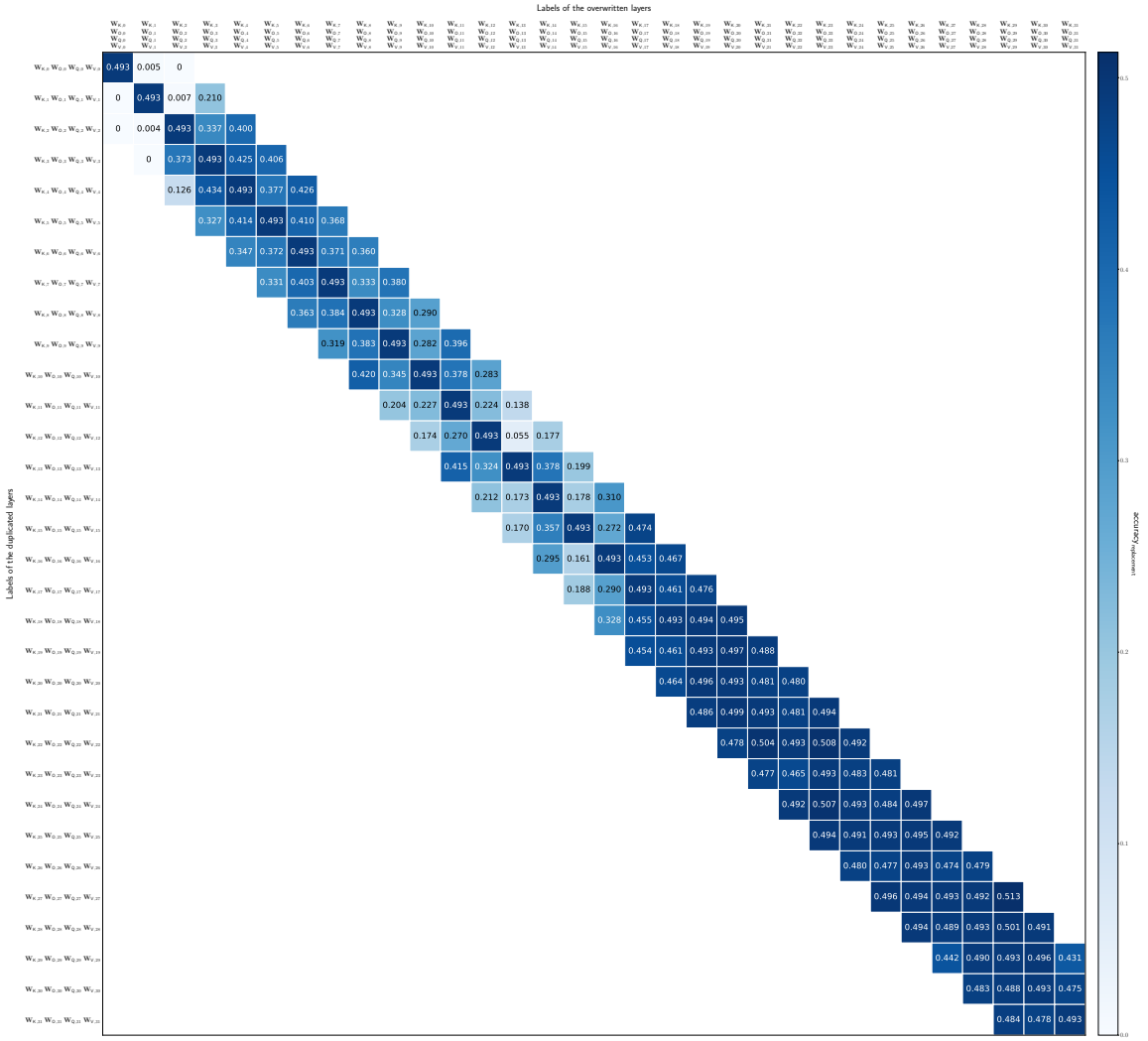


Figure B.4: Heatmap illustrating the accuracy on GSM8K of Llama 3.1 having linear layers of the self-attention module replaced by the ones of a different block.

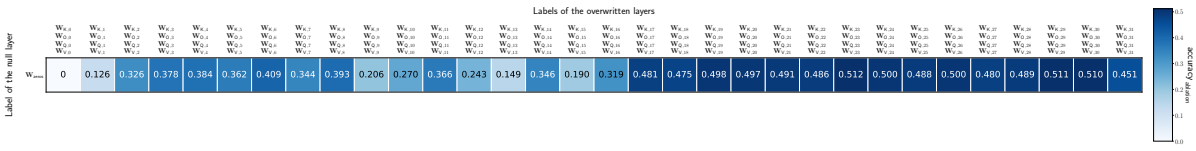


Figure B.5: Heatmap illustrating the accuracy on GSM8K of Llama 3.1 having linear layers of the self-attention module removed.

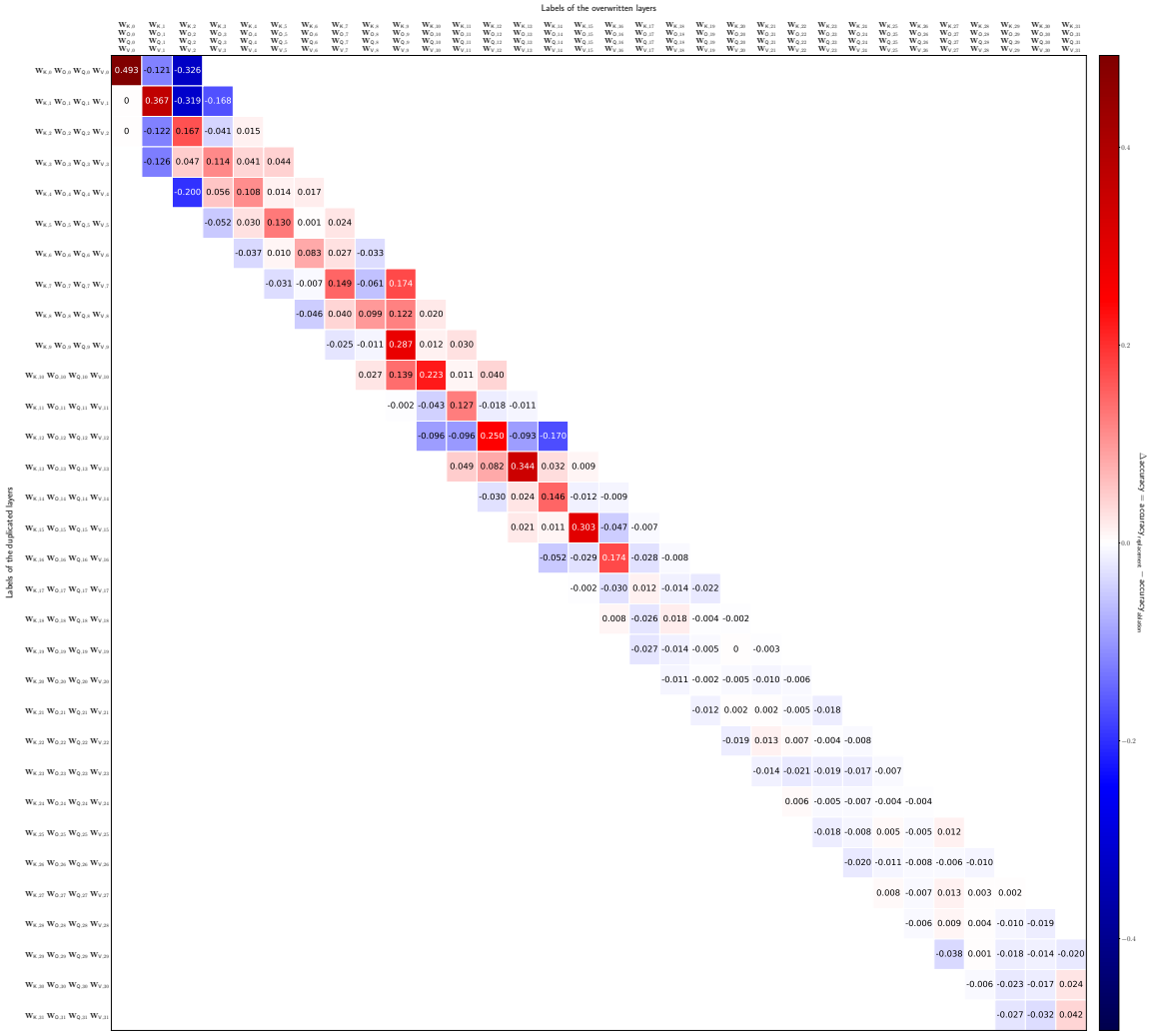


Figure B.6: Heatmap illustrating the difference in performance evaluated on GSM8K between Llama 3.1 with linear layers in the self-attention module replaced by those from another block and Llama 3.1 with the same layers removed.

B.2. Gemma-2-2b

This section investigates the functional redundancy of FFNN and self-attention modules in Gemma 2 (2B) through the replacement analysis. Following the methodology outlined in Section 7.1.3, the results provide insights into the extent to which different layers can be interchanged without significantly impacting model performance.

B.2.1. Characterization of the redundancy in FFNN modules

This section presents the results of the replacement analysis applied to the FFNN modules of Gemma 2 (2B).

Figure B.9 highlights the performance difference on HellaSwag between Gemma 2 (2B) with FFNN layers replaced by those from another block and the model with these layers entirely removed. The heatmap reveals a pronounced negative trend, with most values being negative, apart from a few exceptions. This suggests that substituting FFNN modules with those from a different block significantly degrades model performance, more than removing them. Specifically, in cases where ablation already impacts the model, replacement further exacerbates the degradation, in scenarios where instead ablation has a minor effect, substitution with layers from another block degrade instead the model.

A similar pattern is observed in Figure B.12, where widespread negative values with high absolute magnitudes indicate that replacing FFNNs is even more detrimental on GSM8K compared to HellaSwag.

Despite the overall negative impact given by the substitution of FFNNs, a few exceptions emerge. Specifically, some replacements of the FFNN module in the ninth block (index 8) results in better performance than ablation across both datasets. In the case of GSM8K, where the last layers also partially contribute to solving the task, modules in blocks indexed from 22 to 24 can be substituted with those from certain earlier blocks, leading to improved performance compared to their ablation. This suggests a minor degree of redundancy in the model. However, given the limited improvement relative to module removal and the rarity of such cases, it is insufficient to conclude that redundancy is a general property of FFNN modules in Gemma 2 (2B).

Ultimately, the results of this replacement analysis indicate that using the transformation performed by a FFNN component from one block to approximate another is not effective, implying minimal inter-sub-block redundancy.

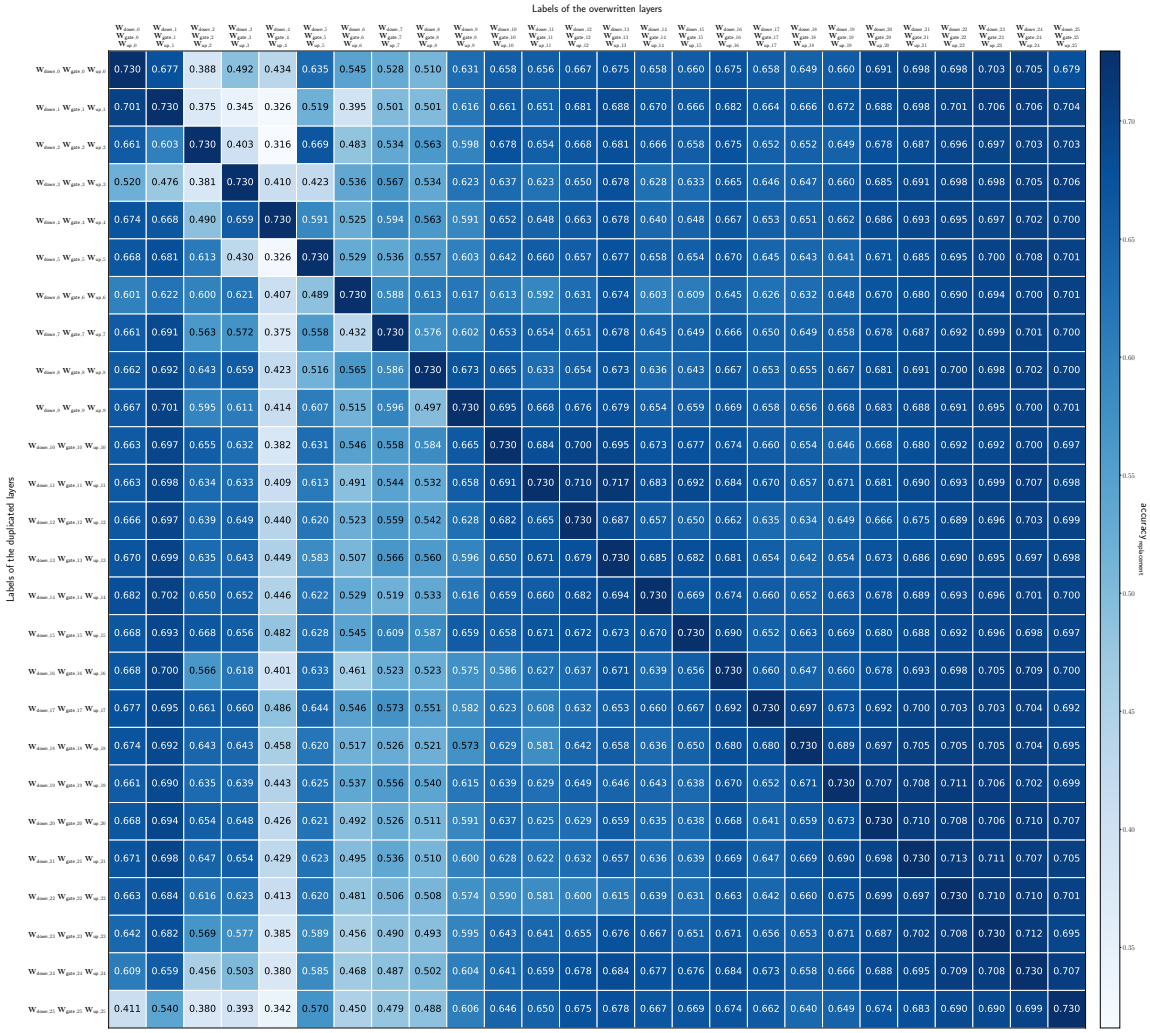


Figure B.7: Heatmap illustrating the accuracy on HellaSwag of Gemma 2 (2B) having linear layers of the FFNN module replaced by the ones of a different block.

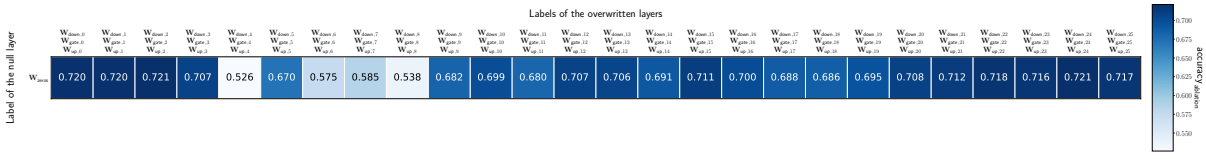


Figure B.8: Heatmap illustrating the accuracy on HellaSwag of Gemma 2 (2B) having linear layers of the FFNN module removed.

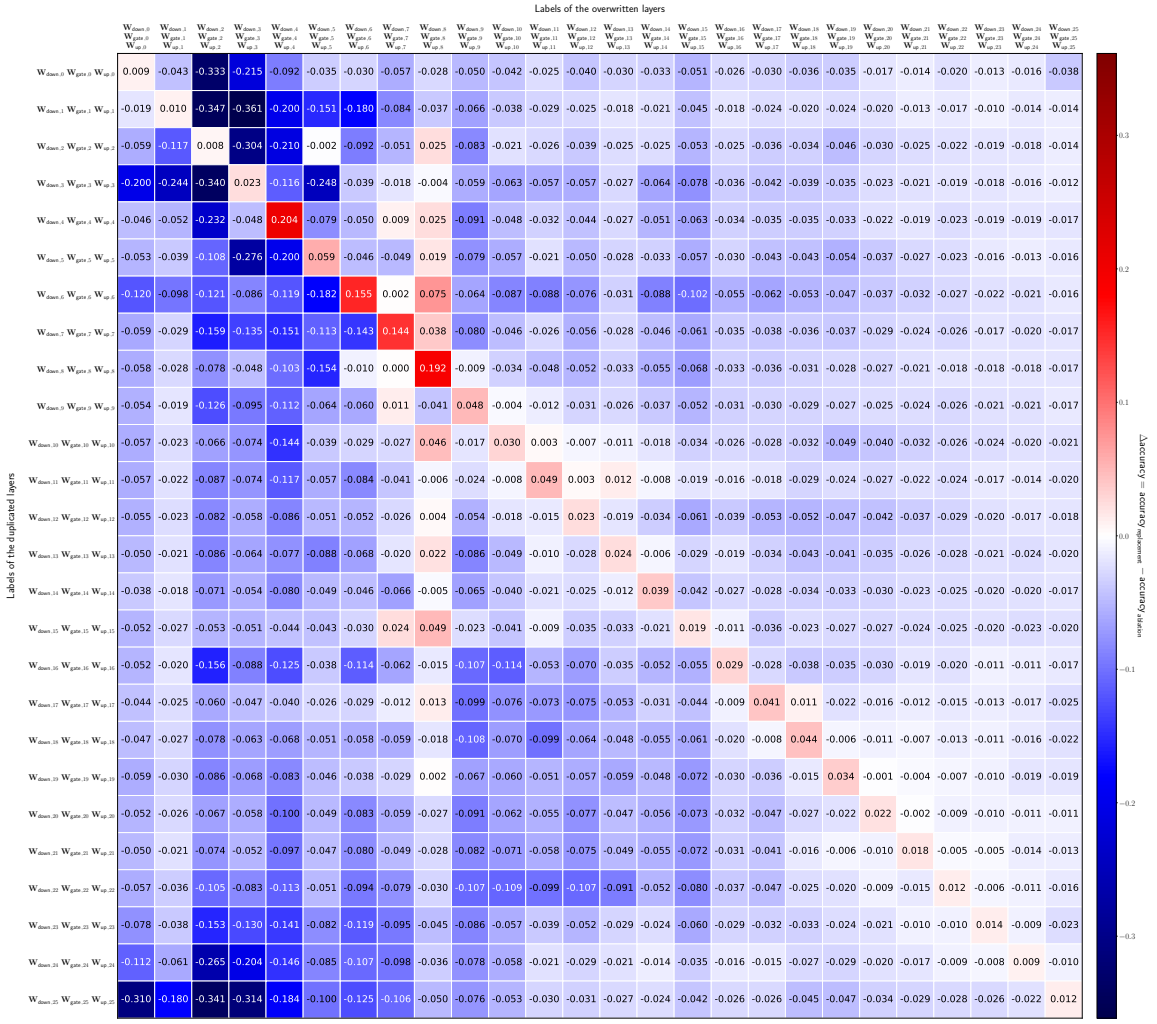


Figure B.9: Heatmap illustrating the difference in performance evaluated on HellaSwag between Gemma 2 (2B) with linear layers in the FFNN module replaced by those from another block and Gemma 2 with the same layers removed.

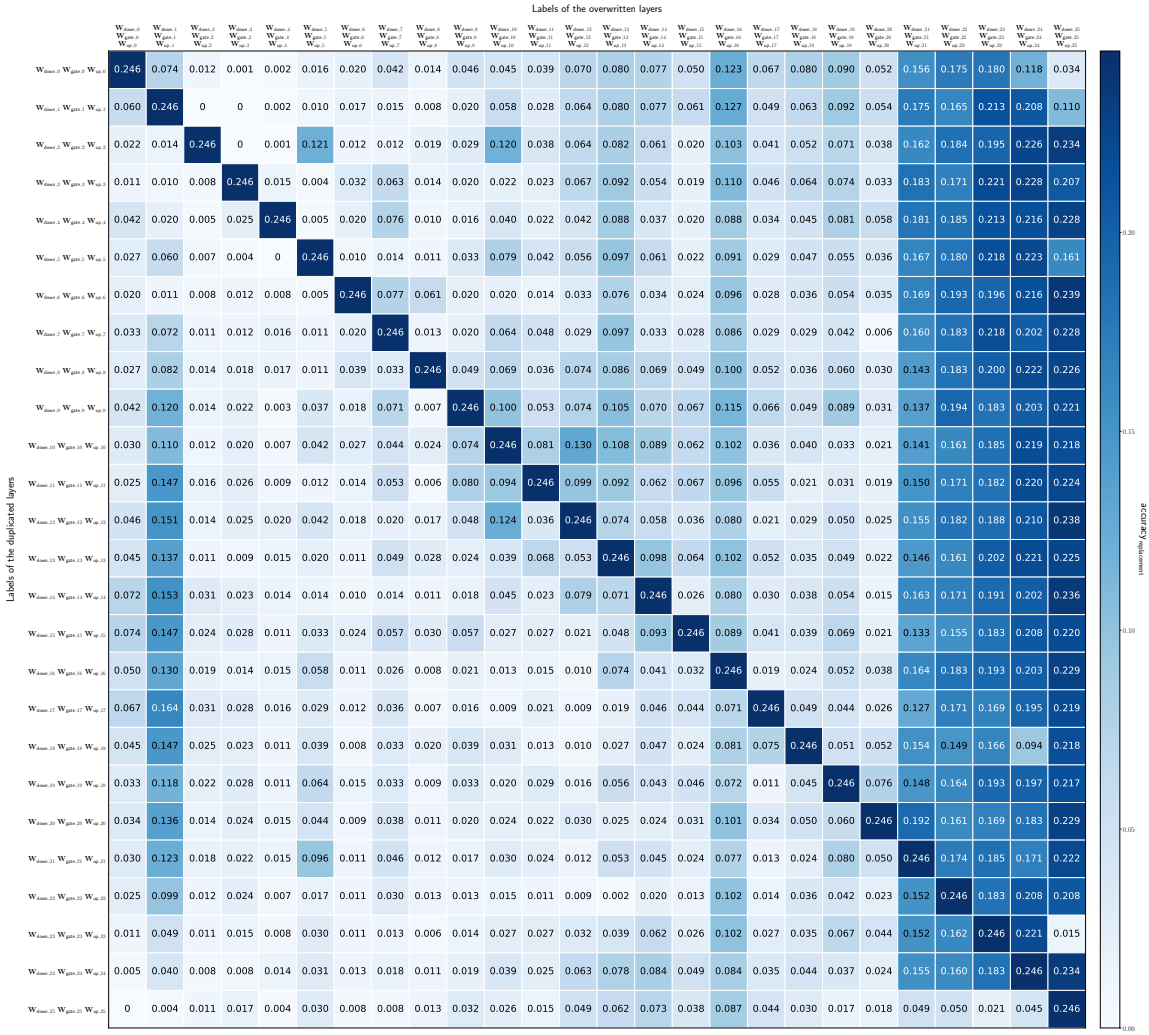


Figure B.10: Heatmap illustrating the accuracy on GSM8K of Gemma 2 (2B) having linear layers of the FFNN module replaced by the ones of a different block.

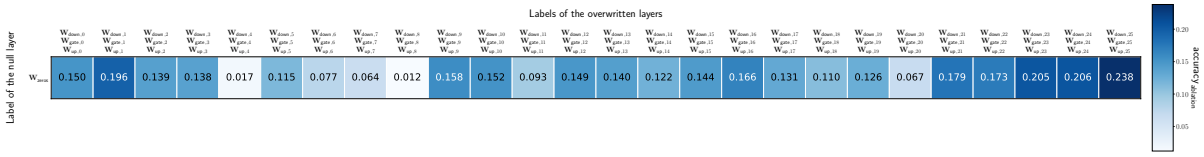


Figure B.11: Heatmap illustrating the accuracy on GSM8K of Gemma 2 (2B) having linear layers of the FFNN module removed.

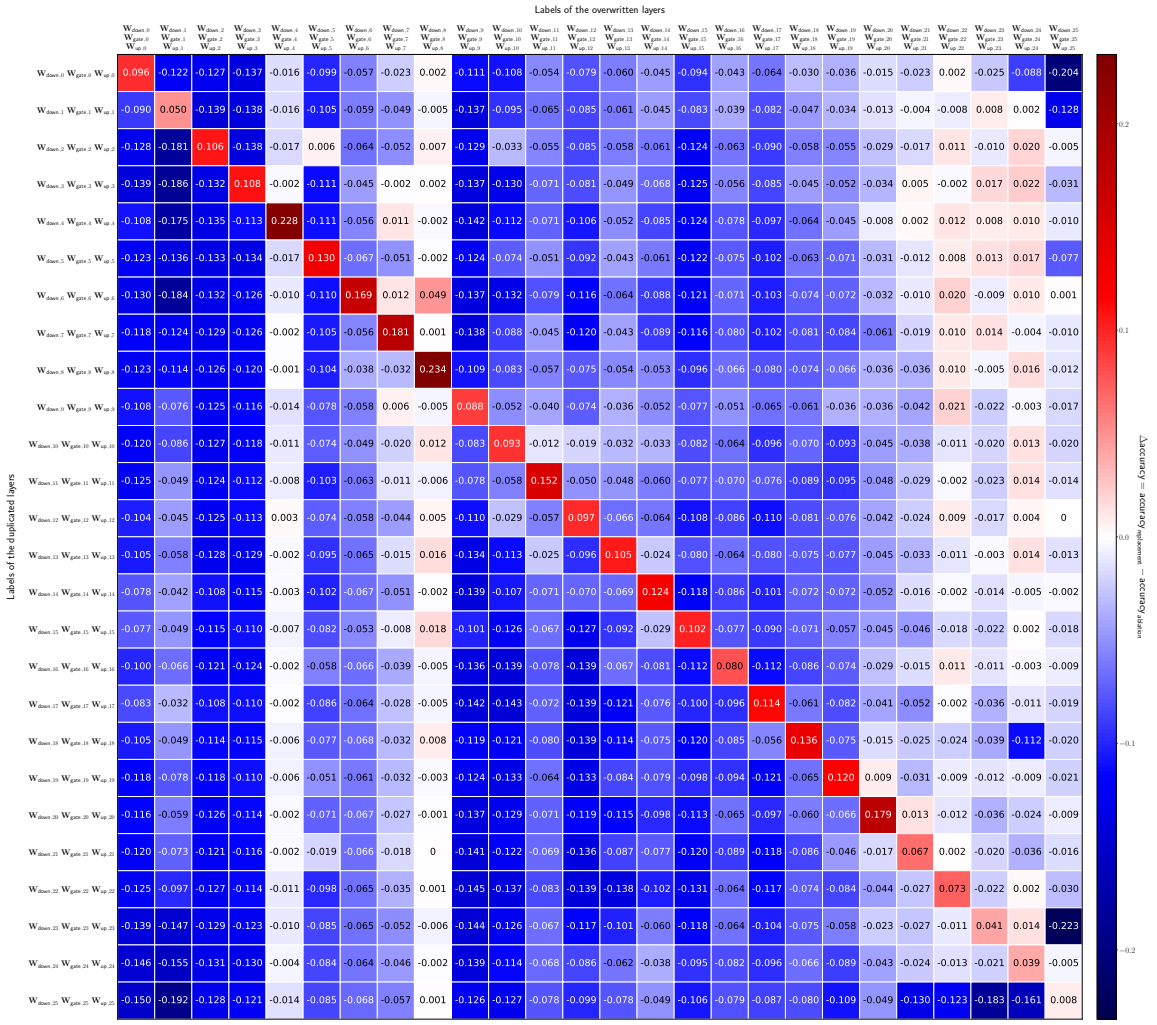


Figure B.12: Heatmap illustrating the difference in performance evaluated on GSM8K between Gemma 2 (2B) with linear layers in the FFNN module replaced by those from another block and Gemma 2 (2B) with the same layers removed.

B.2.2. Characterization of the redundancy in self-attention modules

This section presents the results of the replacement analysis applied to the self-attention modules of Gemma 2 (2B).

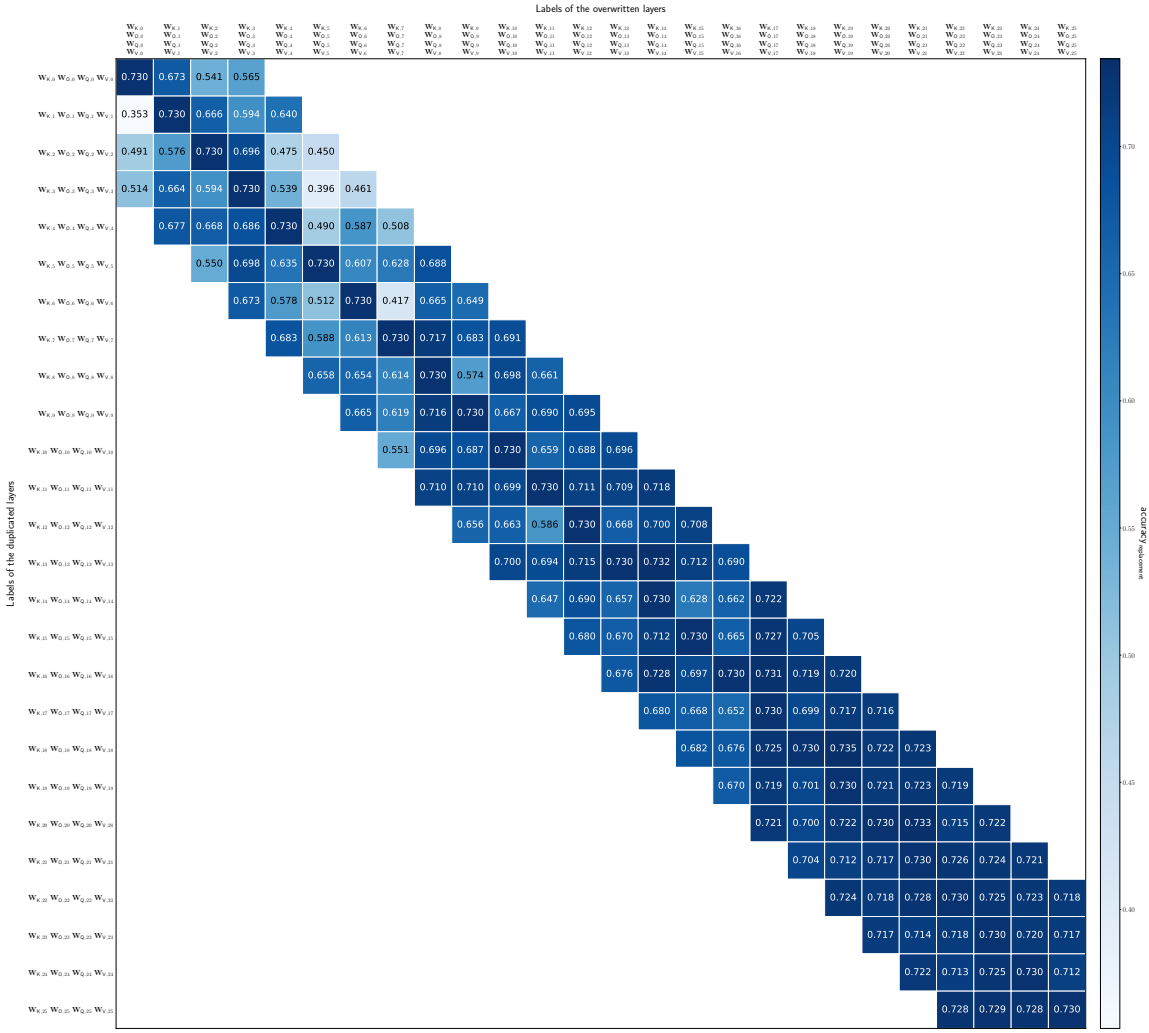


Figure B.13: Heatmap illustrating the accuracy on HellaSwag of Gemma 2 (2B) having linear layers of the self-attention module replaced by the ones of a different block.

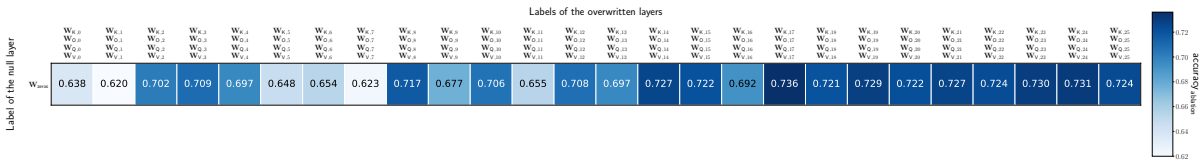


Figure B.14: Heatmap illustrating the accuracy on HellaSwag of Gemma 2 (2B) having linear layers of the self-attention module removed.

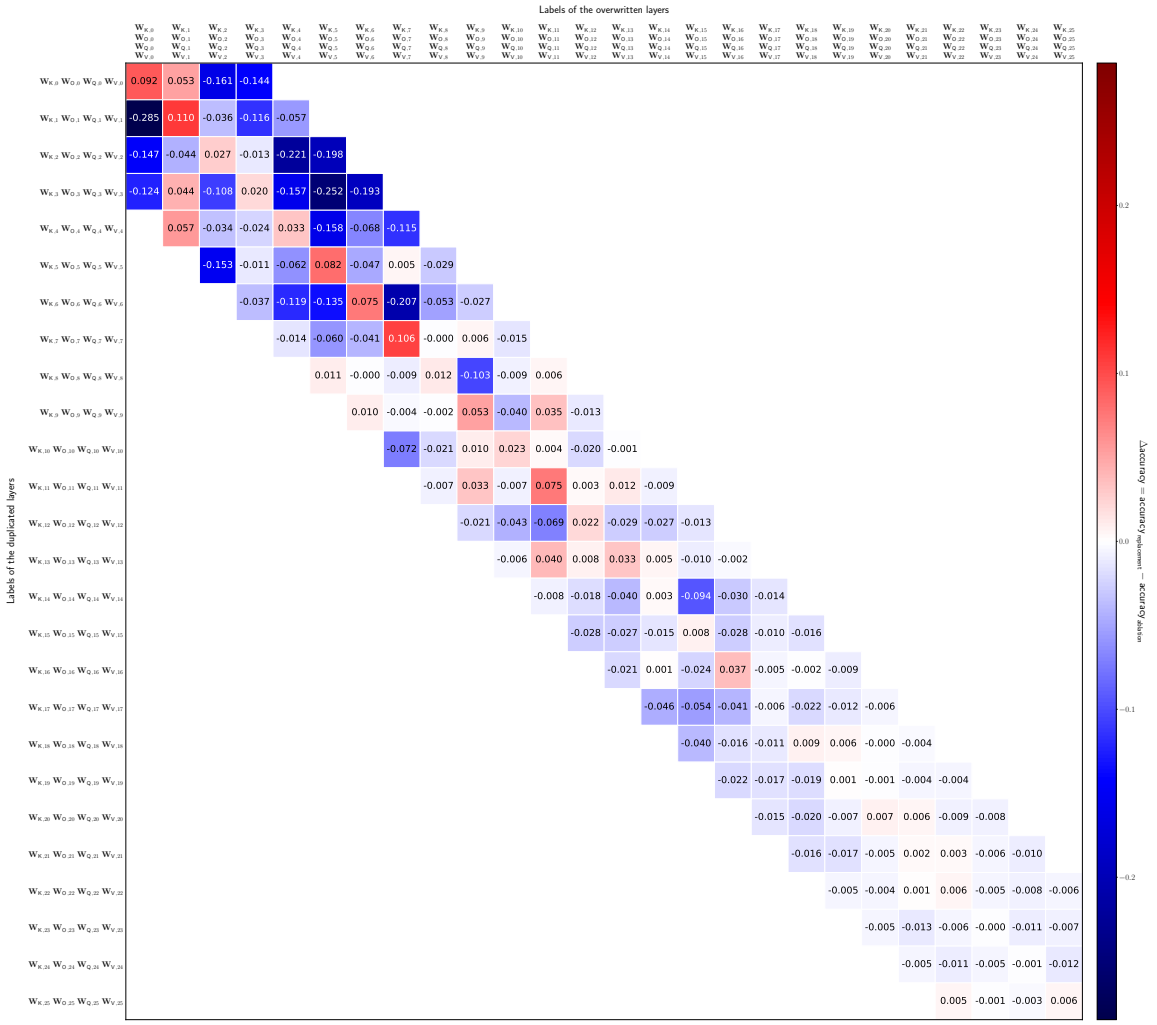


Figure B.15: Heatmap illustrating the difference in performance evaluated on HellaSwag between Gemma 2 (2B) with linear layers in the self-attention module replaced by those from another block and Gemma 2 (2B) with the same layers removed.

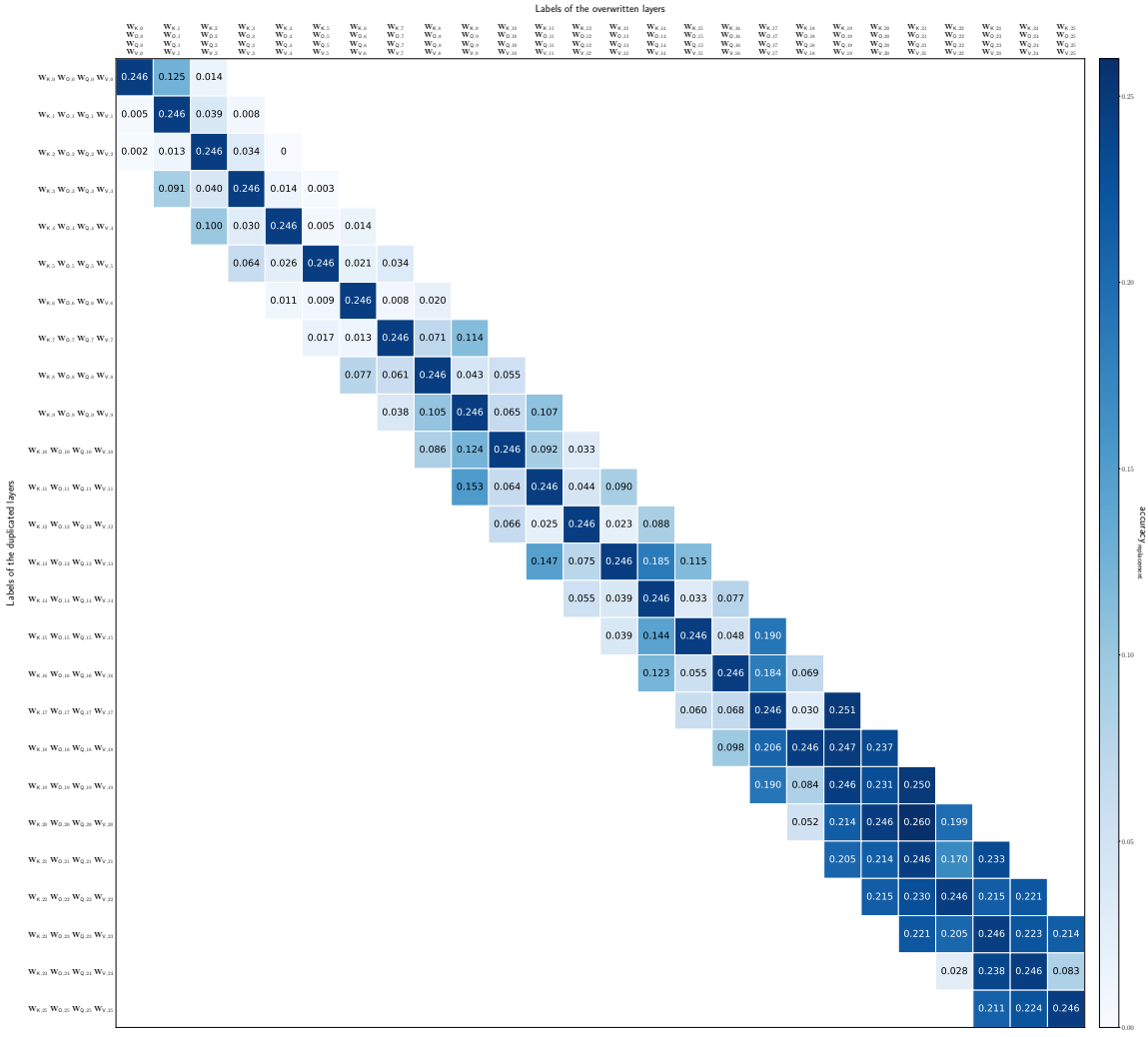


Figure B.16: Heatmap illustrating the accuracy on GSM8K of Gemma 2 (2B) having linear layers of the self-attention module replaced by the ones of a different block.

Figure B.17: Heatmap illustrating the accuracy on GSM8K of Gemma 2 (2B) having linear layers of the self-attention module removed.

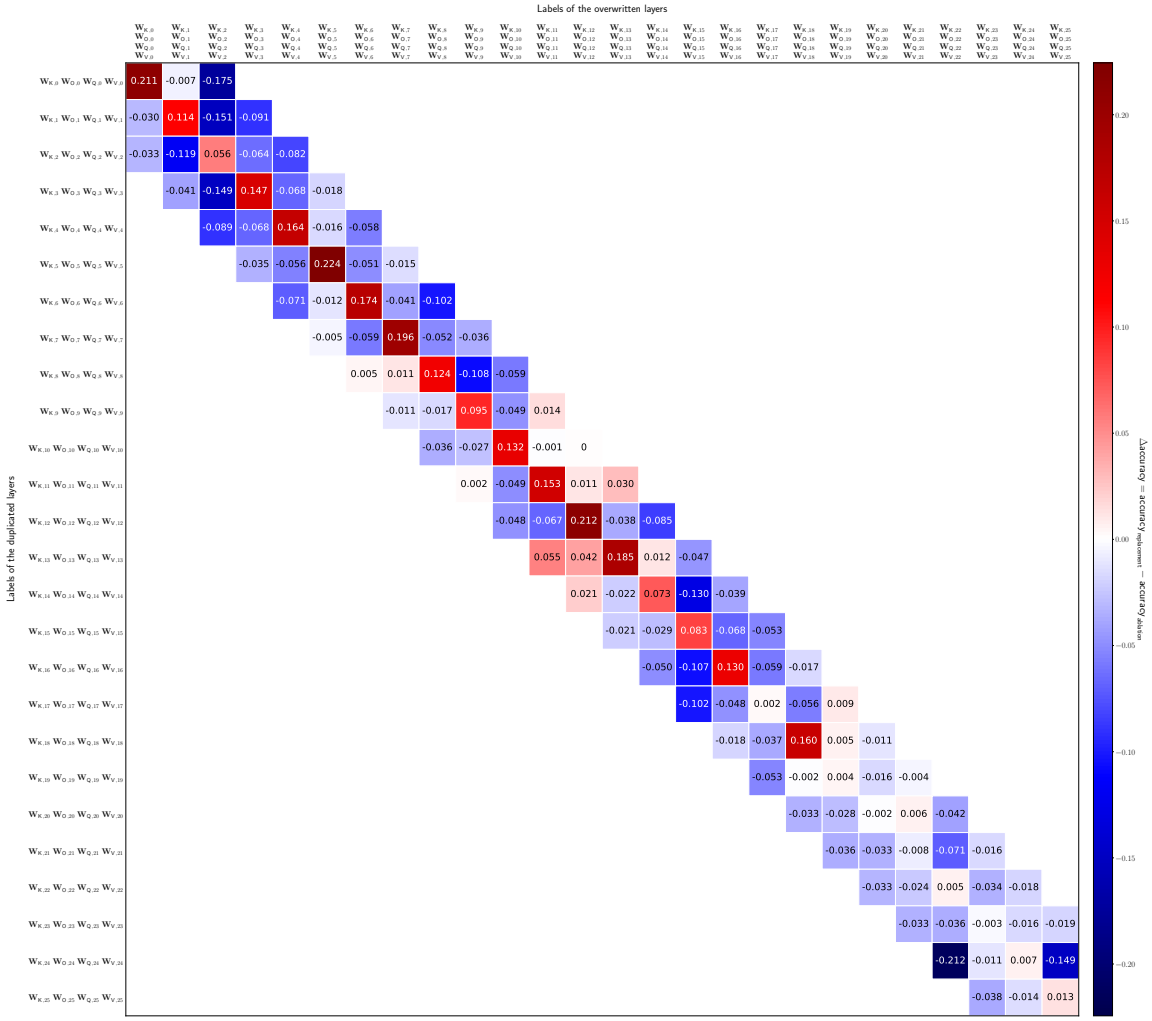


Figure B.18: Heatmap illustrating the difference in performance evaluated on GSM8K between Gemma 2 (2B) with linear layers in the self-attention module replaced by those from another block and Gemma 2 (2B) with the same layers removed.

Figure B.15 highlights the performance difference on HellaSwag between Gemma 2 (2B) with self-attention layers replaced by those from another block and the model with these layers entirely removed. The heatmap reveals a heterogeneous pattern, where most cells indicate that the replacement of the self-attention results in performance degradation comparable to or worse than its ablation. In a few cases, replacement provides slight improvements, primarily in substitutions of the layers within block 1 and in specific configurations of replacements among central layers. A similar trend is observed in Figure B.18, where replacement generally leads to worse performance than removal, with a few

exceptions in mid-block self-attention modules.

The distribution of red cells shows partial consistency in the middle layers across plots, but still functional redundancy appears to be weakly represented in the model.

Ultimately, the replacement analysis indicates that using a self-attention transformation from one block to approximate another is ineffective, reinforcing the notion of minimal inter-sub-block redundancy in Gemma 2 (2B), as previously suggested by the analysis on FFNN layers.

B.3. Comparison between Llama 3.1 and Gemma 2 (2B)

The comparison of experimental results between Llama 3.1 and Gemma 2 (2B) reveals significant differences between the two models. Performance degradation trends are inconsistent across them, with the most critical blocks varying between architectures. Additionally, similarity patterns differ significantly between the two models, suggesting that redundancy, if present, is highly dependent on the specific model architecture.

Gemma 2 (2B) appears to have functional modules that behave more distinctly from one another compared to Llama 3.1. Heatmaps illustrating performance deltas between LLMs with replaced layers and those with ablated layers show a broader and stronger presence of negative values in Gemma 2. In both self-attention and FFNN modules, substituting a module with one from a different block tends to degrade performance more severely than in Llama 3.1, suggesting lower functional redundancy. The reduced redundancy could be attributed to the smaller size of Gemma 2 (2B), which may result in lower over-parameterization and fewer repeated computations across layers.

The inconsistencies observed comparing Gemma 2 and Llama 3.1 highlight significant differences in how concepts are learned and represented across layers in these Transformer-based architectures. The two models differ in terms of knowledge localization and distribution, emphasizing how variations in training processes, architectural modifications, and dataset composition lead to fundamentally different "reasoning patterns" in the resulting models.

C | Appendix C

This appendix provides an overview of the code developed and used to conduct the experiments presented in this thesis. The complete source code is accessible through repositories on GitHub. These repositories contain the implementations of various analytical techniques, compression methods, and auxiliary utilities that form the foundation of the experimental framework.

Comprehensive details on installation, dependency management, and usage of the code are provided in the README file of each repository.

C.1. Experiment-Orchestrator Repository

The Experiment-Orchestrator repository provides the foundational framework for executing experiments. It includes utilities for experiment management and organization, facilitating efficient handling and storage of all relevant information. The framework ensures reproducibility, enhances extensibility, and accelerates the development and validation process.

- `https://github.com/EnricoSimionato/Experiment-Orchestrator.git`

C.2. Redundancy-Hunter Repository

The Redundancy-Hunter repository contains the code to perform redundancy analysis and evaluate the information content of layers in large language models. It builds upon the experimental framework established in Experiment-Orchestrator, adapting it to the specific requirements of these analyses. In addition to the investigations presented in this thesis, it includes further studies conducted during this research that are not explicitly covered in the main text.

- `https://github.com/EnricoSimionato/Redundancy-Hunter.git`

C.3. Alternative-Model-Architectures Repository

The Alternative-Model-Architectures repository implements the compression techniques explored in this study. It provides the necessary classes and utilities to conduct experiments using the proposed methodologies, while also enabling their evaluation. Beyond the experiments presented in this thesis, it also includes additional trials on deep architecture compression.

- `https://github.com/EnricoSimionato/Alternative-Model-Architectures.git`

Acknowledgements

Se sono arrivato fin qui, è merito delle tante persone che ho avuto la fortuna di incontrare in questi intensi ma preziosi anni. A loro va la mia più sincera riconoscenza per il ruolo fondamentale che hanno avuto in questo percorso.

Desidero esprimere un doveroso e sentito ringraziamento al professor Carman e a Vincenzo, senza i quali questa tesi non avrebbe preso forma. Mi hanno accolto nel gruppo di ricerca, dandomi l'opportunità di approfondire temi all'avanguardia e di vivere un'esperienza concreta in ambito accademico.

Durante tutto il periodo di tesi, il professor Carman mi ha offerto numerosi importanti suggerimenti riguardo alle direzioni da seguire, tutti frutto della sua profonda conoscenza della materia. Ho immensamente apprezzato la semplicità e la simpatia con cui ha trasformato ogni incontro in un'opportunità di crescita e confronto.

Un ringraziamento speciale va a Vincenzo. Con pazienza ed esperienza, mi ha offerto un prezioso supporto e le sue idee si sono rivelate fondamentali per completare questo lavoro. Il confronto con lui mi ha permesso di cogliere la passione che mette in ciò che fa, spronandomi a dare il massimo. La sua disponibilità, umanità e simpatia hanno reso questo percorso non solo estremamente formativo, ma anche stimolante e gratificante.

Grazie ai miei genitori e a mia sorella Giulia. Mamma e papà, vi ringrazio troppo raramente per tutto ciò che avete fatto e continuate a fare per noi. Mi avete trasmesso il valore dell'istruzione e del lavoro, insegnandomi che in ogni cosa è fondamentale essere seri, onesti e rispettosi. Senza il vostro supporto e il vostro esempio, non sarei la persona che sono oggi. Giulia, siamo sicuramente diversi e discutere fa parte del gioco, ma ritrovarti a casa è un piacere e rende tutto più divertente.

Un enorme grazie a Maria. Senza di te, con la tua sensibilità e dolcezza, questi anni non sarebbero stati gli stessi e non sarei arrivato dove sono ora. Siamo cresciuti insieme, ne sono profondamente grato. Sei la persona a me più vicina, la persona con cui parlare di tutto e con cui affrontare tutto. Mi credi sempre più bravo di quanto io sia realmente e questo mi sprona a dare ancora di più. Non smetterò mai di ringraziarti.

Grazie ai miei amici di Ronchis, Lorenzo, Simone e Cristian. In questi anni, ogni volta che tornavo a casa, mi hanno fatto sentire come se non fossi mai partito. Siamo amici da tanto tempo e, oltre ai momenti di divertimento e spensieratezza, ho sempre apprezzato l'entusiasmo con cui hanno creduto in me. Ci tengo a ringraziare anche tutti gli altri amici del Friuli, in particolare Giulia e Matilde del gruppo di "studio".

Grazie a Francesco, Michele e Davide. Gli anni a Milano sono iniziati con la nostra partenza verso la grande città e le serate a base di pizza surgelata hanno reso tutto più bello. Grazie anche a Marco e Gabriele per i momenti trascorsi insieme e le epiche, imprevedibili partite di padel.

Durante il mio percorso universitario ho avuto la fortuna di incontrare Alberto e Paolo. Persone straordinarie, capaci di eccellere in tutto ciò che fanno, mi hanno spinto a dare il massimo, negli studi e non solo. Le giornate e le serate tra progetti, pizze, ping pong e Brawl Stars resteranno tra i miei ricordi più belli.