Hidden Properties of Large Language Models

Mingjie Sun

CMU-CS-25-116
May 2025

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:

J. Zico Kolter, Chair
Graham Neubig
Aditi Raghunathan
Kaiming He (Massachusetts Institute of Technology)

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by: the U.S. Air Force under award number FA87501720152; Robert Bosch GMBH under award number 0087016732PCRPO0087023984; Robert Bosch LLC under award number OSP00009188; and the Defense Advanced Research Project Agency under award number HR00112320029. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: deep learning, large language models

Dedicated to my parents, Honglei Hu and Hong Sun

Abstract

Large Language Models (LLMs) are deep learning models trained to understand and generate natural language. Over the course of my PhD, LLMs have profoundly transformed the field of machine learning. They are being actively deployed into numerous commercial products, such as ChatGPT. Moreover, the principles and experiences learned from developing LLMs are still shaping the landscape of machine learning research through paradigms like scaling laws and self-supervised representation learning. However, these rapid advancements may also obscure many fundamental questions about their internal mechanisms and behaviors. As LLM capabilities grow, rigorous scientific investigation beyond conventional training and evaluation workflow is crucial for deeper understanding and continued improvement.

This thesis investigates previously overlooked *'hidden properties'* of LLMs. These hidden properties span the internal weight and activation spaces, as well as their output behaviors. First, we show that LLMs are intrinsically sparse in their weight space. To demonstrate this hidden property, we develop a principled pruning approach that is able to extract effective, sparse sub-networks from pretrained models. Next, we explore the activation space and reveal the existence of structured outliers in LLMs. These activations are extremely few in numbers but exceptionally high in their absolute magnitudes. We call them massive activations. We show that these activations are closely tied to the self-attention mechanism, and propose an alternative attention formulation that is free from such outliers. Finally, we turn to the output space and design a conceptually simple framework to evaluate and study the existence of idiosyncrasies in LLM-generated text. We show that outputs from different models can be distinguished with remarkably high accuracies, and further characterize the specific signatures that underlie these differences. Overall, we hope this thesis can provide an alternative perspective on modern foundation models.

Acknowledgments

My PhD would not have been possible without the support of many people, and I am deeply grateful to all of them.

First and foremost, I would like to express my deepest gratitude to my advisor Zico Kolter. I am incredibly thankful to Zico for admitting me to his group when I was an undergraduate at Tsinghua. I still recall the moment when I received the PhD offer email from him. Over the past six years, he has supported me with constant patience, especially during the first three and a half years of my PhD when I struggled to find my footing and had yet to publish a first-author paper. I vividly remember his congratulation message when my first paper was accepted to CVPR 2023 and that definitely meant a lot to me. His openness and encouragement allowed me the freedom to pursue research questions that truly interested me. I am grateful for the trust and autonomy he has provided, allowing me to grow as a researcher.

I would like to thank all the collaborators and colleagues with whom I've had the privilege to work during my PhD. I am especially grateful to Prof. Zhuang Liu for our research collaboration over the past three years; I have learned so much from him about conducting rigorous research and writing good research papers. I am also deeply thankful to Prof. Graham Neubig, Prof. Aditi Raghunathan, and Prof. Kaiming He for serving on my thesis committee; it has been an honor, as I deeply admire all your work. I am especially grateful to Prof. Aditi Raghunathan, who has served on both my thesis committee and my speaking and writing skills committees. Her guidance and inspiration during my third and fourth years were invaluable in shaping my approach to research. My sincere thanks also go to Sachin Goyal, from whom I learned the true passion for research. I am further grateful to all my other collaborators: Yida Yin, Zhiqiu Xu, Eungyeup Kim, Christina Baek, Xinlei Chen, Anna Bair, Nicholas Carlini, Florian Tramèr, Krishnamurthy (Dj) Dvijotham, Leslie Rice, Hadi Salman, Greg Yang, and Ashish Kapoor.

I am grateful to all current and former members of Locus Lab for providing an enjoyable research environment and sharing diverse perspectives on both research and life. Additionally, I appreciate my speaking and writing skills committee for helping me achieve key milestones in my PhD, as well as my office mates for countless engaging discussions and laughter. I would like to thank my PhD program administrators, Deb Cavlovich and Matthew Stewart, my OIE advisor, Nick Hernandez, and all the administrative staff at CSD for their invaluable support.

I also want to thank those whose friendship supported me during my PhD. I want to thank Yi Zhou for being a wonderful friend and for all the laughter we shared during late-night League of Legends games. I also appreciate Junwen Yang for our enjoyable conversations online and chatting about all sorts of things, even while he was completing his PhD in Singapore.

Last, I want to thank my family – my parents, Honglei Hu and Hong Sun, my brother Mingjun Sun – for always being there to support me. This thesis wouldn't have been possible without your unwavering support throughout my life.

Contents

1 Introduction **1**
1.1 Thesis Organization 2

2 Background **5**
2.1 Definition 5
2.2 Architecture 5
2.3 Training 8
2.4 Inference 9

3 A Simple and Effective Pruning Approach for Large Language Models **11**
3.1 Overview 11
3.2 Introduction 11
3.3 Preliminaries 13
3.4 Approach 13
3.5 Experiments 17
3.5.1 Zero-Shot Tasks 17
3.5.2 Language Modeling 18
3.5.3 Speedup Evaluation 19
3.6 Analysis 20
3.7 Related Work 23
3.8 Additional Results 24
3.8.1 Image Classifiers 24
3.8.2 Previous LLMs 24
3.8.3 Additional Baselines 27
3.8.4 Number of Calibration Samples 28
3.8.5 Robustness Analysis 29
3.8.6 Higher Sparsity 29
3.8.7 Few-shot Results on MMLU 30
3.8.8 Fine-tuning 31
3.8.9 Zero-Shot Tasks 31
3.9 Conclusion 36

4 Massive Activations in Large Language Models **37**
4.1 Overview 37
4.2 Introduction 38
4.3 Massive Activations: What are They? 39
4.3.1 Which Layers? 41
4.3.2 Which Feature and Sequence Dimensions? 41
4.3.3 Difference from Outlier Features 42
4.4 The Role of Massive Activations in LLMs 43
4.5 Effects on Attention 44
4.5.1 Attention is Concentrated on Massive Activations 44
4.5.2 Massive Activations Impose Implicit Attention Biases 46
4.5.3 Explicit Attention Biases Eliminate Massive Activations 48
4.6 Massive Activations in Vision Transformers 49
4.7 Related Work 52
4.8 Additional Results 53
4.8.1 Pretrained LLMs 53
4.8.2 Fine-tuned LLMs 56
4.8.3 BOS Token `<s>` 58
4.8.4 Layer-Level Analysis 60
4.8.5 Attention Concentration on Massive Activations 62
4.8.6 Attention LayerNorm 63
4.8.7 Implicit Attention Biases 65
4.8.8 Explicit Attention Biases 67
4.8.9 Massive Activations in ViTs 68
4.8.10 Registers are Biases in Masked Autoencoders 70
4.8.11 Layer-Level Analysis 70
4.8.12 Models and Datasets 72
4.9 Conclusion and Discussion 74

5 Idiosyncrasies in Large Language Models **75**
5.1 Overview 75
5.2 Introduction 75
5.3 Evaluating Idiosyncrasies in LLMs 77
5.3.1 Main Observations 79
5.3.2 Controlled Experiments 80
5.4 Concrete Idiosyncrasies in LLMs 83
5.4.1 Words and Letters 84
5.4.2 Markdown Formatting 86
5.4.3 Semantics 87
5.5 Implications 90
5.6 Related Work 91
5.7 Additional Results 92
5.7.1 Implementation Details 92
5.7.2 Response Generation 92

5.7.3 Training Setup . . . 92
5.7.4 Prompts for Open-ended Language Analysis . . . 92
5.7.5 Confusion Matrix . . . 93
5.7.6 Words and Letters . . . 94
5.7.7 Characteristic Phrases . . . 94
5.7.8 Unique Markdown Formatting . . . 95
5.7.9 Open-ended Language Analysis . . . 95
5.7.10 Response Demonstrations . . . 97
5.8 Conclusion . . . 111

6 Conclusion **113**
6.1 Future Directions . . . 114

Bibliography **117**

List of Figures

3.1 Illustration of our proposed method Wanda (Pruning by **W**eights **and a**ctivations), compared with the magnitude pruning approach. Given a weight matrix $\mathbf{W}$ and input feature activations $\mathbf{X}$, we compute the weight importance as the elementwise product between the weight magnitude and the norm of input activations ($|\mathbf{W}| \cdot \|\mathbf{X}\|_2$). Weight importance scores are compared on a *per-output* basis (within each row in $\mathbf{W}$), rather than globally across the entire matrix. 12
3.2 Wanda is more robust with less data. 22
3.3 Analysis of comparison groups on pruning image classifiers. 25
3.4 Our pruning metric outperforms the magnitude metric on pruning image classifiers. 25

4.1 **Activation Magnitudes (z-axis) in LLaMA2-7B.** $\mathtt{x}$ and $\mathtt{y}$ axes are sequence and feature dimensions. For this specific model, we observe that activations with massive magnitudes appear in two fixed feature dimensions (1415, 2533), and two types of tokens—the starting token, and the first period (`.`) or newline token (`\n`). 37
4.2 **Massive activations in LLaMA2-13B.** In this model, they appear in two fixed feature dimensions (2100, 4743), and are limited to the starting token. 40
4.3 **Massive activations in Mixtral-8x7B.** In this model, they lie in two feature dimensions (2070, 3398), and are found within the starting token, delimiter tokens and certain word tokens (“`and`” and “`of`”). 40
4.4 Three largest activation magnitudes and the median magnitude at each layer in LLMs. 41
4.5 Attention patterns *before* and *after* massive activations appear in LLaMA2-7B. For each layer, we visualize average attention logits (unnormalized scores before softmax) over all heads, for an input sequence. 45
4.6 Attention patterns *after* massive activations emerge in LLaMA2-13B 45
4.7 Attention patterns *after* massive activations emerge in Phi-2. 46
4.8 Attention LayerNorm and QKV linear projections. 46
4.9 Layer 3, LLaMA2-7B. We highlight the embeddings of the two tokens where massive activations appear: the starting token and the period token. 47
4.10 Value updates from tokens associated with massive activations are essentially the same. 47
4.11 Massive activations disappear when training GPT-2 with explicit attention bias (Equation 4.3). 49

4.12 Three largest activation magnitudes in the output feature of each layer for three GPT-2 models. . . . 49
4.13 Massive activations are present in ViT-L from CLIP and DINOv2, but not MAE. 50
4.14 Three largest activation magnitudes and the median magnitude at each layer in CLIP and DINOv2 ViT-L. . . . 50
4.15 Massive activations in DINOv2-reg ViT-G. . . . 51
4.16 Average attention of the `[CLS]` token. . . . 52
4.17 Massive activations in LLaMA2-70B. . . . 53
4.18 Massive activations in LLaMA3-8B. . . . 54
4.19 Massive activations in LLaMA3-70B. . . . 54
4.20 Massive activations in Phi-2. . . . 54
4.21 Massive activations in Mistral-7B. . . . 54
4.22 Massive activations in MPT-7B. . . . 55
4.23 Massive activations in Falcon-7B. . . . 55
4.24 Massive activations in LLaMA2-7B-Chat. . . . 56
4.25 Massive activations in LLaMA2-13B-Chat. . . . 56
4.26 Massive activations in Mistral-7B-Instruct. . . . 57
4.27 Massive activations in Mixtral-8x7B-Instruct. . . . 57
4.28 Massive activations in LLaMA2-7B when the input is prepended with a BOS token `<s>`. . . . 58
4.29 Massive activations in LLaMA2-13B when the input sequence is prepended with a BOS token `<s>`. . . . 58
4.30 Massive activations in Mixtral-8x7B when the input sequence is prepended with a BOS token `<s>`. . . . 59
4.31 Layer-level analysis of LLMs. For each model, we show the three largest activation magnitudes as well as the median per layer. . . . 61
4.32 Average attention logits over all heads in layers 10, 40 and 60 of LLaMA2-70B. The input sequence is "*This book, including all illustrations and text, is protected under Copyright©2024 and may not be reproduced or transmitted in any form without the prior written permission of the copyright owner.*". . . . 62
4.33 Average attention logits over all heads in layers 3, 10 and 20 of Mistral-7B. The input sequence is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.\n His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*". . . . 62
4.34 Average attention probability over all heads in intermediate layers of Llama2-7B. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.\n His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*". . . . 63
4.35 Average attention probability over all heads in intermediate layers of Mistral-7B. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.\n His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*". . . . 63

4.36 Average attention probability over all heads in intermediate layers of Phi-2. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.*$\backslash n$ *His works include famous plays like 'Romeo and Juliet' and 'Hamlet'*." 64
4.37 Activation trajectory in the attention LayerNorm of LLaMA2-7B and Phi-2, where the LayerNorm input contains massive activations. Note that LLaMA2-7B uses a variant of layer normalization: RMSNorm [176] and Phi-2 uses the default LayerNorm [31]. 64
4.38 Value updates $\sum_{i \in \mathcal{C}} p_i^k v_i$ at layer 3 of LLaMA2-7B, with four input sequences. . 65
4.39 Value updates $\sum_{i \in \mathcal{C}} p_i^k v_i$ at layer 15 of LLaMA2-7B, with four input sequences. 65
4.40 Value updates $\sum_{i \in \mathcal{C}} p_i^k v_i$ at layer 30 of LLaMA2-7B, with four input sequences. 66
4.41 Attention distribution in default GPT-2 and GPT-2 with explicit attention bias. . 67
4.42 Ten largest activation magnitudes at each layer in three GPT-2 models. 68
4.43 Example images. 68
4.44 Illustration of massive activations in CLIP ViT-L for the 4 images shown in Figure 4.43. 69
4.45 Illustration of massive activations in DINOv2 ViT-L for the 4 images shown in Figure 4.43. 69
4.46 Illustration of massive activations in DINOv2 ViT-G for the 4 images shown in Figure 4.43. 69
4.47 Layer-level analysis for ViTs in MAE and CLIP. 70
4.48 Layer-level analysis for ViTs in DINOv2. 71
4.49 Layer-level analysis for ViTs in DINOv2-reg. 71

5.1 **Our framework for studying idiosyncrasies in Large Language Models (LLMs).** We show that each LLM is unique in its expression. In the example shown here on ChatGPT, Claude, Grok, Gemini, and DeepSeek, a neural network classifier is able to distinguish them with a near-perfect 97.1% accuracy. 76
5.2 **Ablations on input length of text embedding models.** Classification accuracies improve as the text embedding models capture more context. Performance begins to saturate beyond an input sequence length of 256. Note that the three lines represent different groups of LLMs and are not directly comparable. 81
5.3 **Different numbers of training samples.** Our sequence classifiers benefit from more training samples. The classification performance converges when using about 10K training samples. 83
5.4 **Example responses from ChatGPT and Claude, showcasing their idiosyncrasies**: characteristic phrases (*left*) and unique markdown formatting (*right*). For clarity, we highlight each characteristic phrase with underline and model-specific color. 84
5.5 **Frequencies of words and letters.** The top 20 most frequently used words of LLMs (left) exhibit distinct patterns for each model, but their letter frequencies (right) are very similar. Results are on the chat API models. 85

5.6 **Characteristic phrases.** We train a logistic regression model on TF-IDF features of chat APIs' outputs and extract the top 10 phrases for each LLM based on the coefficients of these features. We remove common words shared across these LLMs. 86
5.7 **First word.** We analyze the distribution of the first word in chat APIs' responses, with the top 10 most frequent words for each model. These differences in first-word usage explain the non-trivial accuracy with only the first word in Figure 5.2. 87
5.8 **Markdown formatting elements.** Each LLM has a distinctive distribution of markdown formatting elements. 88
5.9 **Results of our open-ended language analysis on ChatGPT and Claude. ChatGPT** features descriptive language, sophisticated markdown formatting, and in-depth details, while **Claude** highlights straightforward tone, minimal structure, and summarized content. 89
5.10 **Inferring model similarity.** We consider 6 LLMs, including 5 chat API models and Phi-4. In each subfigure, we evaluate a five-way classifier on outputs from the excluded LLM and present the distribution of predicted model origins. There is a strong tendency for LLM outputs to be predicted as ChatGPT. 91
5.11 Prompts in our open-ended language analysis. 93
5.12 Confusion matrices for N-way classifiers on three groups of LLMs: chat APIs, instruct LLMs, and base LLMs. 94
5.13 Word and letter frequencies in instruct and base LLMs. 94
5.14 Characteristic phrases for instruct[1]and base LLMs. 95
5.15 Distribution of first word choices in instruct and base LLMs. 95
5.16 Markdown formatting elements for chat APIs (*top*) and instruct LLMs (*bottom*). . 96
5.17 Results of our open-ended language analysis on ChatGPT and **Claude** with different LLM judges. 97
5.18 Additional results of our open-ended language analysis on chat APIs (*top*) and instruct LLMs (*bottom*). 101

List of Tables

3.1 Comparison of Wanda with existing pruning algorithms on LLMs. 16
3.2 Mean zero-shot accuracies (%) of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update. 18
3.3 WikiText perplexity of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update. 19
3.4 Computing the pruning metric of Wanda can be much faster (seconds) than SparseGPT. 20
3.5 Speedup of matrix multiplication (ms) in LLaMA-65B, for structured 2:4 sparsity. 20
3.6 Fine-tuning can mitigate the gap to dense LLM. 20
3.7 Ablation on the pruning configuration. **Bold** results denote the best comparison group for each pruning metric. <u>Underscored</u> results indicate the default pruning configuration of each method. 21
3.8 Effects of the weight update. It offers little or negligible improvement to Wanda. 22
3.9 Pruning Pythia-13B and OPT-13B with various sparsity levels. 26
3.10 Pruning OPT family models with various sizes. 26
3.11 Pruning BLOOM family models with various sizes. 26
3.12 Comparison of pruning *per layer* versus *per output* for OPT models. 27
3.13 Comparison of pruning *per layer* versus *per output* for BLOOM models. 27
3.14 Results for activation pruning methods. 28
3.15 Summary of prior pruning methods on BERT. 28
3.16 Comparisons with prior pruning methods on BERT (unstructured 50% sparsity). . 28
3.17 WikiText validation perplexity of pruned LLaMA and LLaMA-2 under various number of calibration samples, with 50% sparsity. 29
3.18 WikiText validation perplexity of pruned LLaMA and LLaMA-2 models. We report the mean and standard deviation under 5 random seeds. 29
3.19 WikiText validation perplexity of pruned LLaMA and LLaMA-2 models with unstructured 60% sparsity. 30
3.20 WikiText validation perplexity of pruned LLaMA and LLaMA-2 models with unstructured 80% sparsity. 30
3.21 5-shot results (mean accuracies %) on MMLU for unstructured 50% sparsity. . . 31
3.22 The gap between pruned LLMs and dense LLMs can be largely mitigated via fine-tuning. 31

3.23 Accuracies (%) of LLaMA for 7 zero-shot tasks with unstructured 50% sparsity. 32
3.24 Accuracies (%) of LLaMA for 7 zero-shot tasks with 4:8 sparsity. 33
3.25 Accuracies (%) of LLaMA for 7 zero-shot tasks with 2:4 sparsity. 33
3.26 Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with unstructured 50% sparsity. 34
3.27 Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with 4:8 sparsity. 34
3.28 Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with 2:4 sparsity. 35

4.1 Five largest, top 1% and 10%, and the median *activation magnitudes* at a hidden state of three LLMs. The activations that are considered as massive activations are highlighted in bold. 40
4.2 The mean and variance of activation values at several positions, corresponding to the 2 largest, top 1% and 10%, and the median magnitudes within the hidden state. We find that the variation in massive activations is significantly lower in comparison to other activations. 43
4.3 Intervention analysis of massive activations in LLaMA2-7B and 13B. We set massive activations to fixed values and evaluate the perplexity (↓) and zero-shot accuracy (%, ↑) of intervened models. 44
4.4 Intervention analysis of massive activations in CLIP ViT-L. 51
4.5 We fix *all* register features at *every* layer to their means and evaluate the intervened ViTs. 51
4.6 Registers are biases in Masked Autoencoders (MAEs). 70
4.7 Relevant information of LLM models we experimented with in this work. 72
4.8 Relevant information of ViT models we experimented with in this work. 72

5.1 **Classification accuracies for various LLM combinations.** *Top*: results for chat APIs. *Bottom*: results for instruct and base LLMs. Check marks (✓) denote the models included in each combination. We observe high classification accuracies consistently across all model combinations, indicating the presence of distinct idiosyncrasies in LLMs. 78
5.2 **Classification within Qwen2.5 model family.** The classifier can differentiate responses between LLMs within the same model family with reasonably well accuracies. 80
5.3 **Robust generalization to out-of-distribution responses.** We train classifiers on LLM outputs from one prompt dataset and tested on those from another. 80
5.4 **Controlling LLM outputs with prompts.** An instruction is added to the original prompt to specify the output length and format. *Length control* limits responses to one paragraph. *Format control* ensures that responses are in plain text without any format. 81
5.5 **Classifications with different sampling methods.** Distinguishing responses generated by the same model using different sampling strategies is only marginally better than chance accuracy. The results are on Llama3.1-8b instruct model's responses. 82

5.6 **Different sequence embedding models.** LLM2vec achieves the best performance in classifying outputs from various LLMs among the five embedding models we study. . . . 82
5.7 **Text similarity scores.** We evaluate the text similarity of LLM outputs using ROUGE-1, ROUGE-L, and BERTScore. * We follow [267] to rescale BERTScore with respect to the human baseline. The results indicate that responses from different LLMs exhibit low lexical similarity. . . . 83
5.8 **Classifications with only words and letters.** While removing special characters and shuffling words have little impact on accuracies, shuffling letters greatly reduces the performance. . . . 85
5.9 **Classifications with only markdown elements.** Using markdown elements can achieve high accuracies for chat APIs and instruct LLMs, but marginally better results for base LLMs. . . . 86
5.10 **Classifications on rewritten responses.** Paraphrasing or translating LLM outputs achieves an accuracy comparable to that using original counterparts. However, summarizing these texts makes the model less capable of predicting LLM identities. 88
5.11 Our fine-tuning recipe. . . . 93
5.12 Examples of prompt-level interventions on ChatGPT. . . . 98
5.13 Examples of prompt-level interventions on Llama3.1-8b-Instruct. . . . 99
5.14 Examples of only using markdown formatting elements and replacing text content with “xxx” placeholders. . . . 100
5.15 Examples of LLM rewriting using GPT-4o-mini. . . . 102
5.16 Examples from ChatGPT and Claude that illustrate results of our open-ended language analysis in Figure 5.9. . . . 103
5.17 Example response of ChatGPT. . . . 104
5.18 Example response of Claude. . . . 104
5.19 Example response of Grok. . . . 104
5.20 Example response of Gemini. . . . 105
5.21 Example response of DeepSeek . . . 105
5.22 Example response of Llama3.1-8b (instruct). . . . 106
5.23 Example response of Gemma2-9b (instruct). . . . 107
5.24 Example response of Qwen2.5-7b (instruct). . . . 108
5.25 Example response of Mistral-v3-7b (instruct). . . . 109
5.26 Example response of Llama3.1-8b (base). . . . 109
5.27 Example response of Gemma2-9b (base). . . . 109
5.28 Example response of Qwen2.5-7b (base). . . . 109
5.29 Example response of Mistral-v3-7b (base). . . . 110

Chapter 1

Introduction

Large Language Models (LLMs) are a class of machine learning models trained with self-supervision on enormous amounts of text. The origin of LLMs began with the Transformer [1] model introduced in 2017. This model removed recurrence, used purely self-attention, and was originally designed for neural machine translation. In 2019, GPT-2 [2] was released by OpenAI, which brought widespread attention to LLMs. By design, GPT-2 is a decoder-only Transformer and trained with an auto-regressive objective on large corpora. Although small in size compared to today's models, GPT-2 showed strong performance on various tasks, such as translation, summarization and simple question answering. More importantly, its simple design made it easy to scale and to perform inference with prompt-based natural language inputs. Since then, the development of LLMs has followed scaling laws, where larger models trained on more data generally perform better. Later models, such as GPT-3 [3] in 2020 and GPT-4 [4] in 2023, continue to showcase amazing capabilities achievable by LLMs.

To date, LLMs have made a substantial impact on our daily lives. Chatbots based on LLMs, like ChatGPT [5] and DeepSeek [6], are now being widely used all around the world. They have also spurred various applications, such as Cursor and Manus, that are aimed to significantly improve the human workflow. Even more, the goal of achieving Artificial General Intelligence (AGI) feels closer than ever, thanks to ever-increasing performance of LLMs. For researchers in machine learning and deep learning, the success of LLMs has led to a paradigm shift - from training supervised models to large-scale self-supervised representation learning. Another important lesson learnt from the development of LLMs is the importance of scaling laws - simple and scalable methods coupled with more compute and data, which continue to shape progress in AI beyond language domain, e.g., Sora [7] and Whisper [8].

While the impact of LLMs is undeniable, our primary mode of interaction and even much academic research on these models remain frustratingly superficial. In most real-world applications, users interact with these models through black-box APIs – providing a question as input and receiving an answer from APIs or chatbots in return. While this interface is convenient, there is little visibility into the internal processes behind generating these responses. In academic research, much of the focus on LLMs remains at the black-box level, examples include fine-tuning models for better performance [24, 134], developing new prompting strategies [9, 11, 33], and building evaluation benchmarks for new tasks [12, 35]. While these efforts are necessary and important, they offer limited insight into how LLMs actually function. Beyond performance, a scientific understanding of their inner workings and emergent properties is essential for responsible deployment. Yet there still exist many important unanswered questions: Are Transformers truly the right architecture for intelligence? How do LLMs make predictions internally? What feature representations are formed during inference? How do LLMs internally represent concepts, facts, or reasoning chains? As LLMs are increasingly adopted and used in practice, gaining a deeper understanding of their behaviors and inner mechanisms is crucial for long-term progress. Without such understanding, we risk deploying systems with unpredictable failure modes and may struggle to optimize them efficiently, and may lack the principled foundations needed for innovations beyond scaling.

The purpose of this thesis is to conduct a rigorous and scientific investigation into LLMs by uncovering underexplored but critical properties of these models. While most existing studies focus on improving model capabilities, our work seeks to uncover the hidden and complex mechanisms beneath their impressive performance. We show that such investigations can reveal intriguing and unexpected phenomena inherent to LLMs, which we refer to as the hidden properties. We begin by demonstrating, for the first time, that the weight space of LLMs is intrinsically sparse: a significant portion of weight parameters can be removed directly without degrading model performance. Next, we reveal the intriguing presence of structured outliers in the hidden states of large Transformer models, as well as their fundamental connection to the self-attention mechanism. Finally, we analyze the output behaviors of LLMs and identify idiosyncrasies in their generations. Through characterizing these hidden properties, we develop principled tools and methodologies for empirical analysis of LLMs, which we hope can benefit and support future research. We also show that our findings extend beyond the language domain and have practical implications for improving model efficiency and reliability. Altogether, the insights presented in this thesis not only deepen our understanding of LLMs but also help motivate concrete strategies for building next-generation capable models.

1.1 Thesis Organization

In this thesis, we investigate the hidden properties of LLMs from multiple perspectives. Chapter 3 and Chapter 4 focus on the internal representations of LLMs, covering both weights and activations, while Chapter 5 examines distinctive properties of model outputs. Although each chapter targets a different aspect of LLMs, together they aim to provide practical insights for model development and reveal deeper understanding of the models' underlying mechanisms.

The structure of the thesis follows the key building blocks of LLMs, progressing from model parameters (weights), to internal states (activations), and finally to outputs. Below, we briefly summarize the main contributions of each chapter:

Chapter 2 In this chapter, we provide an overview of the background information necessary to understand the subsequent chapters of the thesis. We introduce the definition of Large Language Models (LLMs), then describe the architecture of LLMs, and finally discuss their training and inference procedures.

Chapter 3 This chapter highlights the first hidden property we uncover in LLMs – their weight space is significantly sparser than previously believed. Model pruning is a popular approach for reducing the computation requirements for neural networks, where magnitude pruning has become a well-established popular pruning method. However, prior work has found that the widely used magnitude pruning approach fails catastrophically on pruning LLMs. By carefully designing experiments analyzing the internal representation of LLMs, we identify a key limitation of magnitude pruning: it fails to take into account the emergent outlier features that occur frequently in the activation space of LLMs.

Based on this key insight, we develop a novel, straightforward yet effective pruning method, termed Wanda (*Pruning by Weights and Activations*), designed to induce sparsity directly in pretrained LLMs. Our method prunes weights based on the product of their magnitudes and the corresponding input activations, applied on a *per-output* basis. We show that Wanda significantly outperforms the established baseline of magnitude pruning and performs competitively against prior state-of-the-art method involving intensive weight update. Our results demonstrate, *for the first time*, that effective sparse sub-networks can be obtained directly from pretrained LLMs, revealing that their weight space is inherently sparse.

Chapter 4 This chapter presents our study on a unique type of structured outliers we observe in LLMs' activation space. We show that in the residual hidden states of large Transformers, very few activations exhibit significantly larger values than others (e.g., 100,000 times larger). We call these structured outlier activations *massive activations*. We examine a wide range of popular LLMs and find the widespread existence of massive activations, suggesting that they are an inherent property of LLMs. We answer several important questions on massive activations. First, we show that they appear as constants across most intermediate models, and can be found mostly in fixed feature dimensions, first token and mostly delimiter tokens. Next, we conduct intervention analysis and find that their values largely stay constant regardless of the inputs, and they function as indispensable bias terms in these models.

Furthermore, we find that massive activations are closely connected to an intriguing attention concentration pattern. Specifically, massive activations in the residual hidden states cause attention probabilities in subsequent layers to concentrate heavily on the corresponding tokens. When computing the self-attention output, this leads to the formation of an implicit bias term, created by aggregating value updates from these highly attended tokens. Building on this key insight, we show that several previously proposed fixes for Transformers—such as adding extra learnable tokens (e.g., registers)—can be unified under an alternative attention formulation. Notably, we demonstrate that this alternative formulation is outlier-free: massive activations no longer emerge during pretraining. This is extremely beneficial for low-precision training and inference, which is a widely popular approach for reducing the costs of developing frontier AI models.

Chapter 5 In this chapter, we shift our focus to the output-level behaviors of LLMs, where we aim to characterize differences between models in a novel and principled way. While most recent LLM releases place a heavy emphasis on improvements in benchmark scores, benchmarks alone are not sufficient to capture the nuanced distinctions between models—especially as benchmarks become saturated over time. Our study is driven by the fundamental question: *How can we* *effectively characterize and understand differences in the outputs of various LLMs?*

To address this, we design a conceptually simple classification framework in which a neural network is trained to predict the source model of a given generated text, with each LLM treated as a separate class. We use the accuracy of the classifier as the quantitative indicator of how distinguishable the models' outputs are.

Based on our novel framework, we demonstrate the existence of clear idiosyncrasies in LLMs. We achieve remarkably high classification accuracies by fine-tuning a strong sentence embedding models, revealing that LLM outputs contain strong model-specific signatures. To further analyze these differences, we propose a principled approach that isolates contributing factors through text transformations. Our findings show that these idiosyncrasies are rooted at various levels, including word distributions, formatting styles, and high-level semantics. We conclude the chapter by discussing the broader implications of our findings for the reliable development and evaluation of LLMs.

Chapter 2

Background

In this chapter of the thesis, we will provide an overview of the background information necessary to understand the content of this thesis. We will introduce the basic concepts of what constitutes a Large Language Model (LLM). We will then discuss the process and key elements of building these models, including their architecture, training, and inference procedures.

2.1 Definition

A Large Language Model (LLM) is a machine learning model trained to understand and generate natural language. By its definition, these models are able to take text sequences as inputs and generate natural language outputs. It is worth noting that many of the frontier LLMs [13, 14] are now extended to take multimodal inputs, such as images and audios and also generate multimodal outputs. In this thesis, we will focus on the text-in and text-out LLMs.

LLMs are statistical models that assign a probability distribution to a sequence of input tokens. Tokens represent the basic text units, for example, words and sub-words. We use $x_1, x_2, \dots, x_T$ to denote a sequence of tokens. The LLM predicts the next token by outputing a probability distribution

$$P(x_t|x_1, x_2, \dots, x_{t-1}) \tag{2.1}$$

over the vocabulary of tokens. Due to the chain rule of probability, the joint probability of a sequence of tokens can be expressed as:

$$P(x_1, x_2, \dots, x_T) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots P(x_T|x_1, x_2, \dots, x_{T-1}) \tag{2.2}$$

Note that each term here can be treated as a classification task. This probability is obtained via the Transformer model. In practice, training LLMs is done by maximizing the log-likelihood over a large corpus of the text data. Through this autoregressive training process, the model learns to understand natural language and generate coherent text.

2.2 Architecture

LLMs are based on the Transformer architecture [1], which has driven most of the progress in AI over the past decades. While there are recent advances in language modeling architecture, for instance state-space models [17, 19] and diffusion language models [20, 21], they remain largely at research stage and have not been widely adopted in practice. Therefore, in this thesis, we will focus on the Transformer architecture, which has been a dominant architecture in language modeling ever since its introduction in 2017 [18, 36, 37].

The Transformer architecture is a deep learning architecture based on the self-attention mechanism [38, 39, 40]. The initially proposed Transformer architecture consists of an encoder and a decoder, where the encoder processes the input texts and the decoder generates the output texts. The encoder and decoder are both composed of multiple Transformer blocks, where each block consists of a multi-head self-attention layer and a feed-forward neural network. Due to their autoregressive nature, LLMs discard the encoder part and only keep the decoder part, making it a decoder-only Transformer.

We will now describe the core components of a decoder-only Transformer. First, the input text sequence is turned into a numerical sequence via a tokenizer. These numbers are then encoded via an embedding layer, thus becoming a continuous-number vector embedding. Next, these embedding vectors go through a stack of N Transformer blocks and finally, a linear layer turns the feature embeddings at the last layer into a prediction of the next token.

Tokenizer and Embedding The first layer that the input text passes through is the tokenizer. The tokenizer is responsible for converting the input text into a sequence of tokens, which are then fed into the model. LLMs typically use a sub-word tokenizer, which breaks down words into smaller units, with each unit being a token. Each token is represented by a unique integer. The most commonly used tokenizers in LLMs are based on the Byte Pair Encoding (BPE) [41]. In BPE tokenizer, the vocabulary is built by iteratively merging the most frequent pairs of sub-words in the training corpus. The embedding layer is responsible for converting the tokenized input into a continuous vector embedding. Each token is mapped to a unique vector in a high-dimensional space. One can also view the embedding layer as a look-up table. After the input texts go through the tokenizer and the embedding layer, we have a tensor of shape $B \times T \times D$, where B is the batch size, T is the sequence length and D is the embedding dimension. For now, we will assume that each sequence in the batch has the same length for simplicity. In practice, we can use padding to address imbalanced sequence lengths within a batch.

Multi-Head Self-Attention Multi-Head Self-Attention (MHSA) is a fundamental component of the Transformer architecture. It enables the model to attend to different parts of the input sequence simultaneously, therefore allowing it to efficiently capture long-range dependencies and complex relationships between input tokens.

A Self-Attention Head is based on the scaled dot-product attention mechanism, which computes a weighted sum of the input feature embeddings based on their similarity. Specifically, in a self-attention layer, linear projection layers are first used to transform the input embeddings $X \in \mathbb{R}^{B \times T \times D}$ from the previous layers into three vector embeddings $Q, K, V \in \mathbb{R}^{B \times T \times D_h}$ jointly, where D_h is the head dimension. Then the scaled dot-product attention uses the query and key embeddings to compute the attention scores via the inner product of Q and K^T. A softmax operator is then applied to the dot product to form the attention scores, which are then used to combine the value embeddings into the output. Formally, the scaled dot-product attention is defined as follows:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

Here the output of the attention head is a tensor of the same shape as the query, key and value representations, Q, K and V, which is $B \times T \times D_h$.

It is important to note that in decoder-only Transformers, each token is restricted to attending only to itself and all preceding tokens in the sequence. A causal mask is applied to the attention scores (i.e., the inner product of query and key states), to prevent the model from attending to future tokens. In practice, this can be achieved by masking out the attention scores of the future tokens (e.g., setting them to $-\infty$) before applying the softmax operation.

As the name suggests, multi-head self-attention involves multiple self-attention heads. Each head uses a different linear projection layers to obtain the query, key and value representations. In practice, the outputs of all the self-attention heads are then concatenated together into a single tensor of shape $B \times T \times D$. Then the concatenated embeddings are transformed via a output linear projection. Formally, the multi-head self-attention block is defined as follows: (suppose we have h heads)

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \tag{2.4}$$

where $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{D \times D_h}$ are the linear projection matrices for the i-th head.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{2.5}$$

where $W^O \in \mathbb{R}^{hD_h \times D}$ is the output linear projection matrix. The output of the multi-head self-attention block is a tensor of shape $B \times T \times D$, where D is the embedding dimension. This is the same as the input hidden states X coming from the previous layer.

Feed-Forward Network (FFN) Besides the multi-head self-attention block, each Transformer block also contains a sub-block consisting of a feed-forward network (FFN). Different from self-attention blocks, FFN blocks are applied to each token position separately in parallel. The FFN consists of fully-connected layers with a non-linear activation function, such as GeLU [42].

$$\text{FFN}(x) = \text{GeLU}(XW^{\text{up}})W^{\text{down}} \tag{2.6}$$

where $W^{\text{up}} \in \mathbb{R}^{D \times D_{\text{FFN}}}$ is a up projection layer that projects the feature embedding vectors to a larger dimension D_{FFN} from D. W^{down} is a down projection layer that projects the intermediate FFN hidden states back to the same dimensions as the input tensor X.

It is worth noting that recent LLMs (e.g., Llama [22, 23, 25]) use a variant of the FFN [27], which is defined as follows:

$$\text{FFN}(x) = \big(\text{SiLU}(XW^{\text{gate}}) \cdot XW^{\text{up}}\big)W^{\text{down}} \tag{2.7}$$

where $W^{\text{gate}}, W^{\text{up}} \in \mathbb{R}^{D \times D_{\text{gate}}}$ are two linear projection layers of the same shape. The output of these two linear projection layers are then multiplied in a point-wise manner. Here SiLU is a non-linear activation function proposed in [43]. Mathematically, this activation function is defined as $\text{SiLU}(x) = x \cdot \text{sigmoid}(x)$.

Transformer Layer A transformer layer consists of a multi-head self-attention block and a feed-forward network (FFN). In each sub-blocks, residual connections [30] are layer normalization [31] are applied. Residual connections are the key for building deep neural networks, and layer normalization is used to improve the training performance and stability. Formally, the transformer layer is defined as follows:

$$X' = X + \text{MHSA}(\text{LayerNorm}(X)) \tag{2.8}$$

$$\text{TransformerLayer}(X) = X' + \text{FFN}(\text{LayerNorm}(X')) \tag{2.9}$$

where X' is the intermediate hidden state produced by the self-attention block. Since the layer normalization is applied to the input of each block (self-attention and FFN), this is also called a pre-norm Transformer layer.

Linear Head The final layer of the LLM is a linear classification head, which is used to predict the next token in the input sequence. The linear head takes the output of the last transformer block and applies a linear transformation to project it onto the vocabulary space. Therefore, the weight matrix of the linear head has the shape of $D \times V$, where V is the size of the vocabulary in the tokenizer. Then the predicted probabilities over the entire vocabulary are obtained via the softmax operation, from which we can sample the next token.

2.3 Training

We describe the training procedure of LLMs. The training process consists of two main steps: pretraining and post-training.

Pretraining Pretraining is the process of training the LLM model on a large corpus of text data. The goal of pretraining is to make the model understand natural language. This process is often unsupervised, meaning that there is no need for labeled data of any forms. Through unsupervised pretraining, the model is able to learn meaningful representations from the pre-training datasets. Popular pretraining paradigms include masked language modeling [18] and autoregressive language modeling [2].

LLMs as decoder-only Transformers use autoregressive pretraining, where the model is trained to predict the next token in a sequence given the preceding tokens. Specifically, at each token position, the model computes a probability distribution of the next token, which is a vector of the vocabulary size. Then the cross-entropy loss is computed between the predicted probability distribution and the ground truth next token. This loss is averaged over all the tokens in the training corpus. During pretraining, the weight parameters of LLMs are updated via back-propagation to minimize this average cross-entropy loss. Formally, the autoregressive pretraining objective is defined as follows:

$$\mathcal{L}_{\text{pretrain}} = -\frac{1}{T}\sum_{t=1}^{T} \log P(x_t | x_1, x_2, \ldots, x_{t-1}) \tag{2.10}$$

for a sequence of tokens $x_1, x_2, \ldots, x_T$ in the training corpus.

Post-Training Post-training is the process of refining and optimizing the pretrained LLMs. While pretraining LLMs learn general language representations, pretrained LLMs are not able to follow human instructions and produce coherent texts. Therefore, in order for these models to be useful in practice, post-training is a critical process that aligns these pretrained models with human preferences.

There are two popular post-training paradigms: supervised fine-tuning and the alignment stage. Supervised fine-tuning (SFT) is the process of adapting pretrained LLMs to specific tasks or domains by training them on curated datasets of interest. For instance, one can fine-tune a pretrained LLM on dialogue data for it to become a helpful assistant. SFT is typically done via supervised learning, meaning that the training data consists of input-output pairs. However, in many cases, labeled data is resource intensive to collect. What's more, supervised fine-tuning is not able to generalize well outside the training domain. In practice, SFT is often used in conjunction with alignment approaches.

The alignment stage aims to adapt LLMs to better match human preferences. Typically, this is done via reinforcement learning (RL) fine-tuning to learn from feedback data, such as reinforcement learning from human feedbacks (RLHF) [26] and reinforcement learning from AI feedbacks (RLAIF) [29]. The model is optimized via reward signals that reflect the quality of their own generations. This reinforcement learning process encourages the model to produce outputs that are more helpful and aligned with human values.

2.4 Inference

We describe the inference procedure of LLMs. It typically consists of two main steps: pre-filling and decoding.

Pre-filling This is the stage where the model processes the entire input prompt before autoregressive decoding. In the pre-fill stage, the model receives the full prompt and passes all the input tokens through the network in a single forward pass. No prediction is performed in this stage. During pre-filling, the key and value states of the input tokens at each Transformer layer are cached for the decoding stage. These key and value states are also known as KV cache, which allows the model to reuse key and value states of existing processed tokens in the sequence without recomputation.

Decoding Decoding is the process of generating the output sequence token by token in an autoregressive fashion. At each token prediction, the model predicts the next token by sampling from the predicted probability distribution over the entire vocabulary. There can be various sampling strategies. One common strategy is greedy decoding, a deterministic process where the token with the highest probability is always selected as the next token. Another popular method is temperature sampling, which uses a temperature parameter to control the randomness of the probability distribution: a higher temperature means more random outputs, while a lower temperature makes the output more deterministic.

Chapter 3

A Simple and Effective Pruning Approach for Large Language Models

3.1 Overview

As their size increases, Large Languages Models (LLMs) are natural candidates for network pruning methods: approaches that drop a subset of network weights while striving to preserve performance. Existing methods, however, require either retraining, which is rarely affordable for billion-scale LLMs, or solving a weight reconstruction problem reliant on second-order information, which may also be computationally expensive. In this paper, we introduce a novel, straightforward yet effective pruning method, termed Wanda (Pruning by **W**eights **and a**ctivations), designed to induce sparsity in pretrained LLMs. Motivated by the recent observation of emergent large magnitude features in LLMs, our approach prunes weights with the smallest magnitudes multiplied by the corresponding input activations, on a *per-output* basis. Notably, Wanda requires *no* retraining or weight update, and the pruned LLM can be used *as is*. We conduct a thorough evaluation of our method Wanda on LLaMA and LLaMA-2 across various language benchmarks. Wanda significantly outperforms the established baseline of magnitude pruning and performs competitively against recent method involving intensive weight update. Code is available at `https://github.com/locuslab/wanda`.

3.2 Introduction

Large language models [3, 4] have recently reshaped the field of NLP with their remarkable performance across a range of complex language benchmarks [86, 105, 107]. However, these models, with their billions of parameters, usually require significant computational resources. To democratize LLMs, considerable efforts have been taken to mitigate their high computational cost. Many of the notable advancements to date have centered on model quantization, a process where parameters are quantized into lower bit-level representations. The fast pace of LLM quantization research [57, 58, 87, 154] has led to substantial resource savings for these models [111, 150].

Network pruning [66, 77, 78], on the other hand, shrinks network sizes by removing specific weights from the model – essentially setting them to zero. Along with quantization, it is often

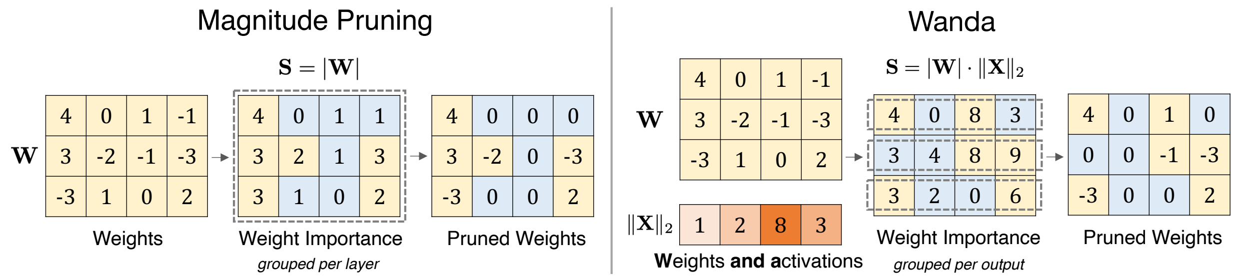


Figure 3.1: Illustration of our proposed method Wanda (Pruning by **W**eights **and a**ctivations), compared with the magnitude pruning approach. Given a weight matrix $\mathbf{W}$ and input feature activations $\mathbf{X}$, we compute the weight importance as the elementwise product between the weight magnitude and the norm of input activations ($|\mathbf{W}| \cdot \|\mathbf{X}\|_2$). Weight importance scores are compared on a *per-output* basis (within each row in $\mathbf{W}$), rather than globally across the entire matrix.

considered another popular approach for compressing neural networks. However, it has received relatively little focus in compressing LLMs. This seems to contradict the trend of model compression in the pre-LLM era, where both approaches have received large amounts of research effort. A quick review of existing pruning methods reveals a possible reason: they typically require retraining [74, 95], training from random initializations [67, 70, 79] or even an extensive iterative process [69, 80]. The sheer amount of computational resources required by LLMs limits these methods. A recent LLM pruning approach, SparseGPT [59], does not require traditional retraining, but still demands a computationally intensive weight update process.

The argument concerning the need for retraining and weight update does not fully capture the challenges of pruning LLMs. One might reasonably expect to obtain a fairly high-performing initialization point for retraining using existing popular pruning methods. However, a recent study [59] finds that magnitude pruning [66], a well-established pruning approach, fails dramatically on LLMs even with relatively low levels of sparsity. Considering the past success of magnitude pruning on smaller networks, this result suggests that LLMs, despite having 100 to 1000 times more parameters, are substantially more difficult to prune directly.

In this work, we address this challenge by introducing a straightforward and effective approach, termed Wanda (Pruning by **W**eights **and a**ctivations). This technique successfully prunes LLMs to high degrees of sparsity without *any* need for modifying the remaining weights. We are motivated by an observation from a recent study [58], where a small subset of hidden state features are exceptionally large in magnitude, a property unique to LLMs. We find that augmenting the standard weight magnitude pruning metric with the input activations, is surprisingly effective as a measure for evaluating the weight importance. Specifically, we introduce a novel pruning metric, where each weight is evaluated by the product of its magnitude and the norm of the corresponding input activations, estimated using a small set of calibration data. Our method uses this metric to induce sparsity in pretrained LLMs by comparing weights locally within each output of linear layers and removing lower priority weights. Our approach is computationally efficient, able to be executed in a single forward pass, and requires minimal memory overhead.

We empirically evaluate Wanda on the widely adopted LLaMA [22] and LLaMA-2 [23] model families. Our results demonstrate Wanda can find efficient sparse networks from pretrained LLMs, without any retraining or weight update. Our approach Wanda outperforms the standard magnitude pruning by a large margin and also competes favorably with the prior best LLM pruning method [59], while requiring a lower computational cost. We hope our work serves as a baseline for future work in this area, and encourages further exploration in understanding sparsity in LLMs.

3.3 Preliminaries

Magnitude Pruning [66] is a standard pruning technique to induce sparsity in neural networks. It removes individual weights based on their magnitudes, where weights with magnitudes below a certain threshold are removed. In practice, this threshold is typically determined by comparing weights locally within each layer or globally across the whole network. Despite its simplicity, magnitude pruning has been used to find extremely sparse networks [69] and now stands out as a strong baseline approach [95] for neural network sparsification.

Emergent Large Magnitude Features have been observed in Transformer-based large language models. [58] discover that once LLMs reach a certain scale (in practice, around 6B parameters), a small set of hidden state features emerges with significantly larger magnitudes than the remaining ones. These outlier features exhibit several intriguing characteristics. First, they have very large magnitudes, about 100 times larger than typical hidden state values. Second, they are usually sparse and exist in certain feature dimensions. Finally, these outlier features are essential for the predictive capability of LLMs: zeroing out these features at inference time results in significant degradation of language modeling performance.

3.4 Approach

In this section, we motivate and describe our pruning method, Wanda (Pruning by **W**eights **and** **a**ctivations), which consists of two simple but essential components. First, we propose a novel pruning metric that incorporates both weights and input activations into the computation of weight importance. Second, we compare weights on a *per-output* basis instead of across the whole layer, which we find is crucial for pruning LLMs effectively. An overview of Wanda is shown in Figure 3.1.

A Motivating Example. Consider a neuron with two inputs and corresponding weights: $\mathbf{y} = \mathbf{w}_1\mathbf{x}_1 + \mathbf{w}_2\mathbf{x}_2$, where $|\mathbf{w}_1| \leq |\mathbf{w}_2|$. Now suppose the goal is to select one weight for removal while incurring less change on the output. The standard approach of magnitude pruning would always remove weight $\mathbf{w}_1$, which may be a good strategy if input features $\mathbf{x}_1$ and $\mathbf{x}_2$ have similar magnitudes. However, as recently observed in LLMs [58], the two input features can differ significantly in scale. For instance, it is possible that $|\mathbf{x}_1| \gg |\mathbf{x}_2|$, and as a result, $|\mathbf{w}_1\mathbf{x}_1| \gg |\mathbf{w}_2\mathbf{x}_2|$. In this case, we should remove weight $\mathbf{w}_2$ instead, because this removal clearly exerts a smaller influence on the neuron output $\mathbf{y}$ than removing weight $\mathbf{w}_1$.

This motivating example with the simplest linear layer hints at a major limitation of magnitude pruning: it does not take into account input activations, which could play an equally important role as weight magnitudes in determining the neuron output. For pruning LLMs, this is especially critical considering the emergent large magnitude features found within them. Thus, as the first part of our method, we propose a pruning metric designed explicitly for LLMs to handle such a limitation, while also maintaining the simplicity of magnitude pruning.

Pruning Metric. Consider a linear layer with weight $\mathbf{W}$ of shape $(C_{\texttt{out}}, C_{\texttt{in}})$. For language models, this linear layer takes in input activations $\mathbf{X}$ with a shape of $(N \times L, C_{\texttt{in}})$, where N and L are batch and sequence dimensions respectively. For each individual weight, we propose to evaluate its importance by the product of its magnitude and the corresponding input feature norm. Specifically, the score for the current weight $\mathbf{W}_{ij}$ is defined by:

$$\mathbf{S}_{ij} = |\mathbf{W}_{ij}| \cdot \|\mathbf{X}_j\|_2 \tag{3.1}$$

where $|\cdot|$ represents the absolute value operator, $\|\mathbf{X}_j\|_2$ evaluates the ℓ_2 norm of jth features aggregated across $N \times L$ different tokens, and the final score is computed by the product of these two scalar values. We find that ℓ_2 norm tends to work better than other norm functions (e.g., ℓ_1 and ℓ_∞) in measuring activation magnitudes. This is possibly because ℓ_2 norm is generally a smoother metric.

This metric is interesting in several aspects. First, when the input channel of the considered weight has large magnitude features, the weight itself tends to be assigned a larger importance score even if it has a low magnitude. This tackles the problem we encounter in the motivating example. The effect can be seen in Figure 3.1, where weights corresponding to the large magnitude feature are more likely to be preserved with Wanda. Second, its computation is straightforward. Once we obtain the norm vector of input feature activations, the weight importance can be calculated using an element-wise dot product. Last, we find empirically that this metric is robust and can be easily estimated using a modest number of calibration samples, without access to the original training data.

Comparison Group. Generally, in a pruning method, each weight is first assigned an importance score, such as the pruning metric we discussed above. These weights are then grouped into *comparison groups* where weights within each group are compared against one another. Within each comparison group, weights with lower importance scores are pruned. Most previous pruning methods default to comparing weights locally within each layer or globally across the whole network.

While layer-wise and whole-network comparisons have been the popular options, we find that pruning LLMs could benefit from a more localized grouping. In our method, we compare and remove weights on a *per-output* basis (per row in Figure 3.1), where weight importance scores are compared locally within each output neuron. Specifically, for a weight $\mathbf{W}_{ij}$ that connects input j to output i inside the linear layer, we define the *comparison group* for this weight as all weights connecting to output i:

$$\mathbf{G}_{ij} = \{\mathbf{W}_{uv} \mid u = i\} \tag{3.2}$$

Algorithm 1 PyTorch code for Wanda

```
# W: weight matrix (C_out, C_in);
# X: input matrix (N * L, C_in);
# s: desired sparsity, between 0 and 1;

def prune(W, X, s):
  metric = W.abs() * X.norm(p=2, dim=0)

  _, sorted_idx = torch.sort(metric, dim=1)
  pruned_idx = sorted_idx[:,:int(C_in * s)]
  W.scatter_(dim=1, index=pruned_idx, src=0)
  return W
```

Under this comparison group, for a pre-defined sparsity ratio $s\%$, we eliminate $s\%$ of the weights connected to *each output*. This practice may seem counter-intuitive, since we are basically pruning under a stricter sparsity pattern. However, we find that it is *consistently better* than layer-wise pruning for LLMs. Notably, this holds true not only for our proposed pruning metric (Equation 3.1) but also the standard magnitude metric. This shows that maintaining a balanced pruning ratio across output features is important for pruning LLMs effectively.

To see if the superiority of pruning *per output* over *per layer* holds true in general, we conduct additional experiments on pruning image classifiers. However, we do not observe similar trend in image classification models, suggesting that our observations regarding pruning *per output* might be unique to LLMs. We hope this intriguing observation encourages practitioners to be more cautious in choosing the comparison group.

Procedure. Wanda can be implemented and integrated seamlessly within a *single* forward pass of the LLM model, where feature norm statistics $\|\mathbf{X}_j\|_2$ are estimated with a set of calibration data. We provide the PyTorch code of our approach in Algorithm 1. Given a pretrained LLM, we compute our pruning metric from the initial to the final layers of the network. After pruning a preceding layer, the subsequent layer receives updated input activations, obtained on the pruned weights of the previous layer. Then the pruning metrics are computed. A recent method for pruning LLMs, SparseGPT [59], requires a sophisticated weight update procedure in an iterative pruning process, while Wanda does not induce any additional weight update.

Structured N:M Sparsity. While Wanda so far has been developed for unstructured sparsity, it can be easily extended to structured N:M sparsity [139]. Structured N:M sparsity requires that at most N out of every M contiguous weights to be non-zero. It can leverage NVIDIA's sparse tensor cores to accelerate matrix multiplication in practice. Wanda can be naturally extended to structured N:M sparsity, where we compare weights using the same metric among every M consecutive weights, for all weights connected to an output.

Method	Weight Update	Calibration Data	Pruning Metric $\mathbf{S}_{ij}$	Complexity
Magnitude	✗	✗	$\lvert\mathbf{W}_{ij}\rvert$	$O(1)$
SparseGPT	✓	✓	$\left[\lvert\mathbf{W}\rvert^2/\text{diag}\left[(\mathbf{X}\mathbf{X}^T+\lambda\mathbf{I})^{-1}\right]\right]_{ij}$	$O(d^3_{\texttt{hidden}})$
Wanda	✗	✓	$\lvert\mathbf{W}_{ij}\rvert\cdot\lVert\mathbf{X}_j\rVert_2$	$O(d^2_{\texttt{hidden}})$

Table 3.1: Comparison of Wanda with existing pruning algorithms on LLMs.

Remark. We discuss the connection between Wanda and a few existing works. SparseGPT formalizes the problem of pruning LLMs by solving a local layer-wise reconstruction problem, where their pruning metric and weight update procedure is inspired from Optimal Brain Surgeon (OBS) [78]. The pruning metric in SparseGPT is:

$$\mathbf{S}_{ij} = \left[\lvert\mathbf{W}\rvert^2/\text{diag}\big((\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I})^{-1}\big)\right]_{ij} \tag{3.3}$$

Here $\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}$ in the denominator is the Hessian $\mathbf{H}$ for the layer-wise reconstruction problem and λ is the Hessian dampening factor to avoid the collapse of inverse computation. With careful inspection, we observe that our metric in Equation 3.1 is similar to the above when λ is 0 and only the diagonal elements of the Hessian matrix $\mathbf{X}^T\mathbf{X}+\lambda\mathbf{I}$ are retained. Starting from the pruning metric in Equation 3.3, we show the exact reduction steps and corresponding reduction conditions as follows:

$$\mathbf{S}_{ij} \overset{\lambda=0}{=} \left[\lvert\mathbf{W}\rvert^2/\text{diag}\left((\mathbf{X}^T\mathbf{X})^{-1}\right)\right]_{ij} \underset{\text{approx.}}{\overset{\text{diagonal}}{=}} \left[\lvert\mathbf{W}\rvert^2/\left(\text{diag}(\mathbf{X}^T\mathbf{X})\right)^{-1}\right]_{ij} = \left(\lvert\mathbf{W}_{ij}\rvert\cdot\lVert\mathbf{X}_j\rVert_2\right)^2 \tag{3.4}$$

In the last reduction step, the diagonal of $\mathbf{X}^T\mathbf{X}$ is $\text{diag}(\lVert\mathbf{X}_j\rVert_2^2)$, and thus the denominator can be simplified to $(\lVert\mathbf{X}_j\rVert_2^2)^{-1}$. The resulting metric in Equation 3.4 is the square of our proposed metric. This simplification substantially reduces the required computation of weight importance, eliminating the need for computing any matrix inverses.

In the 1980s, [77] have set up a pioneering framework for neural network pruning named Optimal Brain Damage (OBD). It uses second-order information without off-diagonal elements in Hessians for faster approximation. Later, Optimal Brain Surgeon (OBS) develops upon OBD partly by taking into account the off-diagonal elements. Wanda can be seen as a renaissance of OBD – it may be viewed as applying a process similar to OBD to each neuron, with *local* output reconstruction as the objective function, whereas the original OBD uses the *global* training objective. This is analogous to the relationship between SparseGPT and OBS.

A comparison of LLM pruning methods can be found in Table 3.1. Computing the pruning metric of Wanda has a reduced time complexity compared to SparseGPT, because it does not involve inverse computation. Overall, our method Wanda (Pruning by **W**eights **and a**ctivations) has several attractive properties as an approach for pruning LLMs:

1. It maintains the simplicity of magnitude pruning in the pre-LLM era, requiring no gradient computation via back-propagation or any second-order Hessian inverses, but is also highly effective in discovering sparse networks in pretrained LLMs.

2. Wanda can be done with a single forward pass of the LLM. At each layer, the pruned weights can be decided in one shot without an iterative procedure. In practice, computing the pruning metric of Wanda can be 300 times faster in pruning LLMs compared with SparseGPT.
3. Unlike SparseGPT, our approach entails no weight update on pruned networks, suggesting that LLMs have effective sparse sub-networks that are *exact*, instead of them merely existing in the neighborhood of the original weights.

3.5 Experiments

Models and Evaluation. We evaluate Wanda on the two most widely adopted LLM model families: LLaMA 7B/13B/30B/65B [22] and LLaMA-2 7B/13B/70B [23] (LLaMA-2 34B is not released). Results for prior LLM families can be found in Chapter 3.8.2. We measure the performance of pruned models on zero-shot tasks and language modeling. For zero-shot evaluation, we use seven tasks from EleutherAI LM Harness [52]. Following previous works on LLM compression [57, 59], we evaluate the perplexity on the held-out WikiText [63] validation set.

Baselines. We compare Wanda with two prior pruning approaches. Magnitude pruning [66] is a simple and strong baseline in which weights are discarded based on their magnitudes. SparseGPT [59] is a second-order pruning method for LLMs, based on solving a layer-wise reconstruction problem. In Chapter 3.8.3, we compare with additional pruning methods.

Both Wanda and SparseGPT require calibration data to estimate input statistics (see Table 3.1). To control this variable factor, we use the *exact same* set of calibration data as SparseGPT, which consists of 128 sequences with context length size sampled from C4 training set [36]. In Chapter 3.8.4, we provide additional analysis on the number of calibration samples.

Sparsity. For all pruning methods, we focus on pruning the linear layers (skipping the first embedding layer and the final classification head), which account for around 99% of the total LLM parameters. We impose a uniform sparsity for all linear layers. We evaluate three types of sparsity: unstructured sparsity, structured 4:8 and 2:4 sparsities. The magnitude pruning baseline is extended to structured N:M sparsity in a similar spirit to our method, as described in the previous section.

3.5.1 Zero-Shot Tasks

Comparison with Baselines. In Table 3.2, we show the mean zero-shot accuracies on 7 zero-shot tasks of pruned LLaMA and LLaMA-2 models. We refer the reader to Chapter 3.8 for task-wise performance. Across both unstructured and structured sparsities, Wanda outperforms the well-established magnitude pruning approach by a large margin, while also rivaling the previous best approach SparseGPT. Given that no fine-tuning takes place, there is a noticeable gap between sparse pruned LLMs and the original dense LLMs. However, as the model size

Method	Weight Update	Sparsity	LLaMA 7B	LLaMA 13B	LLaMA 30B	LLaMA 65B	LLaMA-2 7B	LLaMA-2 13B	LLaMA-2 70B
Dense	-	0%	59.99	62.59	65.38	66.97	59.71	63.03	67.08
Magnitude	✗	50%	46.94	47.61	53.83	62.74	51.14	52.85	60.93
SparseGPT	✓	50%	**54.94**	58.61	63.09	66.30	**56.24**	60.72	**67.28**
Wanda	✗	50%	54.21	**59.33**	**63.60**	**66.67**	**56.24**	**60.83**	67.03
Magnitude	✗	4:8	46.03	50.53	53.53	62.17	50.64	52.81	60.28
SparseGPT	✓	4:8	**52.80**	55.99	60.79	64.87	**53.80**	**59.15**	65.84
Wanda	✗	4:8	52.76	**56.09**	**61.00**	**64.97**	52.49	58.75	**66.06**
Magnitude	✗	2:4	44.73	48.00	53.16	61.28	45.58	49.89	59.95
SparseGPT	✓	2:4	**50.60**	**53.22**	58.91	62.57	**50.94**	54.86	63.89
Wanda	✗	2:4	48.53	52.30	**59.21**	**62.84**	48.75	**55.03**	**64.14**

Table 3.2: Mean zero-shot accuracies (%) of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update.

increases, this accuracy gap diminishes. Remarkably, unstructured 50% sparse LLaMA-65B and LLaMA-2-70B is able to match the zero-shot accuracies of their dense counterparts.

Large Sparse *vs.* Small Dense. It might be of interest to some readers on the comparison between large sparse LLMs and small dense LLMs with similar parameter counts. For zero-shot performance, we find the trend differs across the types of sparsity. For unstructured sparsity, large sparse LLMs are often better than small dense LLMs on zero-shot performance: unstructured 50% sparse LLaMA-65B (66.67%) outperforms dense LLaMA-30B (65.38%); unstructured 50% sparse LLaMA-2-13B (60.83%) outperforms dense LLaMA-7B (59.71%). Intriguingly, this gap is much larger for few-shot tasks (see Chapter 3.8). For structured sparsity, the trend is reversed: without any fine-tuning, large sparse LLMs have worse zero-shot performance than small dense LLMs in general.

3.5.2 Language Modeling

In Table 3.3, we report the perplexity of pruned LLaMA and LLaMA-2 models. For robustness analysis under random sampling of the calibration data, see Chapter 3.8.

Without any weight update, Wanda outperforms the established pruning approach of magnitude pruning by a large margin. For instance, for LLaMA-7B, Wanda is able to find sparse networks with a perplexity of 7.26, significantly better than the magnitude pruning baseline 17.29. This result suggests that exact and effective sparse sub-networks exist for LLMs. For unstructured 50% sparsity, Wanda performs on par with the prior best approach SparseGPT. We provide results for higher sparsity levels (60% and 80%) in Chapter 3.8. The comparison between Wanda and SparseGPT is mixed for structured sparsity. On smaller models (e.g., 7B), SparseGPT outperforms

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
Magnitude	✗	50%	17.29	20.21	7.54	5.90	14.89	6.37	4.98
SparseGPT	✓	50%	**7.22**	6.21	5.31	**4.57**	6.51	5.63	**3.98**
Wanda	✗	50%	7.26	**6.15**	**5.24**	**4.57**	**6.42**	**5.56**	**3.98**
Magnitude	✗	4:8	16.84	13.84	7.62	6.36	16.48	6.76	5.54
SparseGPT	✓	4:8	8.61	**7.40**	6.17	5.38	8.12	6.60	4.59
Wanda	✗	4:8	**8.57**	**7.40**	**5.97**	**5.30**	**7.97**	**6.55**	**4.47**
Magnitude	✗	2:4	42.13	18.37	9.10	7.11	54.59	8.33	6.33
SparseGPT	✓	2:4	**11.00**	**9.11**	7.16	6.28	**10.17**	8.32	5.40
Wanda	✗	2:4	11.53	9.58	**6.90**	**6.25**	11.02	**8.27**	**5.16**

Table 3.3: WikiText perplexity of pruned LLaMA and LLaMA-2 models. Wanda performs competitively against prior best method SparseGPT, without introducing any weight update.

Wanda on 2:4 sparsity. Wanda is more favorable for larger models, e.g., LLaMA-30B (2:4 and 4:8) and LLaMA-2-70B (2:4).

3.5.3 Speedup Evaluation

Pruning Speed. The theoretical computational complexity of Wanda is lower than SparseGPT (Table 3.1). Here we compare their empirical pruning speed. Specifically, we measure the accumulated time for computing the pruning metric at each layer (excluding the forward pass process shared by both methods) on NVIDIA A6000 GPUs. Results are shown in Table 3.4. Wanda incurs negligible time overhead relative to SparseGPT. The fast speed of Wanda is particularly useful when pruning needs to be performed on a real-time basis, e.g., training sparse models from scratch [153] and finding the optimal sparsity [124].

Inference Speed. We evaluate the inference speedup for structured 2:4 sparsity on NVIDIA A6000 GPUs. Following the evaluation setup of [59], we measure the latency of matrix multiplication in linear layers. We perform simulation analysis using the high-performance GEMM kernel in NVIDIA CUTLASS library. Results for LLaMA-65B (batch size of 1) can be found in Table 3.5. Structured 2:4 sparsity is able to bring notable inference speedup (around 1.6×) for linear layers in LLMs. For end to end latency, we observe a speedup of 1.24× on LLaMA-7B (251ms as compared to 312ms). Last, we emphasize that the inference speedup is not unique to our pruning method but is delivered by the inherent power of sparsity for speeding up computation.

	LLaMA			
Method	7B	13B	30B	65B
SparseGPT	203.1	339.0	810.3	1353.4
Wanda	**0.54**	**0.91**	**2.9**	**5.6**

Table 3.4: Computing the pruning metric of Wanda can be much faster (seconds) than SparseGPT.

LLaMA Layer	Dense	2:4	Speedup
`q/k/v/o_proj`	3.49	2.14	1.63×
`up/gate_proj`	9.82	6.10	1.61×
`down_proj`	9.92	6.45	1.54×

Table 3.5: Speedup of matrix multiplication (ms) in LLaMA-65B, for structured 2:4 sparsity.

3.6 Analysis

We study several aspects of Wanda to better understand its effectiveness in pruning LLMs. We use the LLaMA-7B model and prune to unstructured 50% sparsity, unless otherwise specified.

Fine-tuning. We study how fine-tuning could recover the performance drop of pruned LLMs, as observed in the previous section. We investigate two strategies for fine-tuning LLMs: LoRA [134] fine-tuning and full parameter dense fine-tuning. Fine-tuning is conducted on C4 training dataset and the objective is the pre-training auto-regressive loss. The pruned mask is kept fixed during fine-tuning. We fine-tune pruned LLaMA-7B with all three types of sparsities: unstructured 50%, structured 4:8 and 2:4. Table 3.6 summarizes the results for mean zero-shot accuracies and perplexity after fine-tuning Wanda pruned LLaMA-7B models. See Chapter 3.8 for task-wise performance.

LoRA Fine-tuning. We enforce a limited computational budget (1 GPU and 12 hours). The low rank ($r = 8$) adapter is applied on the query and value projection matrices in attention layers. For LLaMA-7B, LoRA introduces only around 0.06% additional parameters, leaving the total sparsity

Evaluation	Dense	Fine-tuning	50%	4:8	2:4
Zero-Shot	59.99	✗	54.21	52.76	48.53
		LoRA	56.53	54.87	54.46
		Full	**58.15**	**56.65**	**56.19**
Perplexity	5.68	✗	7.26	8.57	11.53
		LoRA	6.84	7.29	8.24
		Full	**5.98**	**6.63**	**7.02**

Table 3.6: Fine-tuning can mitigate the gap to dense LLM.

level still around 50%. With LoRA fine-tuning, we are able to restore the performance of pruned LLMs by a non-trivial amount. One notable instance is that LoRA fine-tuning improves the zero-shot performance of structured 2:4 sparse LLaMA-7B from 48.53% to 54.46%, outperforming the original unstrucutred 50% sparse LLaMA-7B (54.21%).

Full Parameter Fine-tuning. We conduct full parameter dense fine-tuning. We enforce a limited computational budget (4 GPU and 3 days). Compared to LoRA fine-tuning, full parameter dense fine-tuning is able to mitigate the gap between pruned LLMs and dense LLMs even further. For unstructured 50% sparsity, full parameter fine-tuning could improve pruned LLaMA-7B from 54.21% to 58.15% in terms of zero-shot accuracy, close to that of dense LLaMA-7B (59.99%).

Pruning Configuration. Wanda differs from previous methods in both the pruning metric and the comparison group. We conduct ablation experiments to better understand their impact. The three pruning metrics can be found in Table 3.1. SparseGPT adopts a local comparison group inside a layer, where weights connected to 128 consecutive *input* channels form a group. Wanda groups weights connected with a single *output* channel. Therefore, we ablate two blocksize options (128 and 1) and the input/output choice. For simplicity, we use (input/output, blocksize) to denote each local comparison group, e.g., (input, 1). For this experiment, we do not perform the weight update procedure in SparseGPT to focus on the pruning configuration.

	Comparison Group				
Pruning Metric	layer	(input, 1)	(input, 128)	(output, 1)	(output, 128)
Magnitude: $\lvert\mathbf{W}_{ij}\rvert$	<u>17.29</u>	**8.86**	16.82	13.41	17.47
SparseGPT: $\left[\lvert\mathbf{W}\rvert^2/\mathrm{diag}(\mathbf{H}^{-1})\right]_{ij}$	7.91	8.86	<u>8.02</u>	**7.41**	7.74
Wanda: $\lvert\mathbf{W}_{ij}\rvert \cdot \lVert\mathbf{X}_j\rVert$	7.95	8.86	8.12	<u>**7.26**</u>	7.71

Table 3.7: Ablation on the pruning configuration. **Bold** results denote the best comparison group for each pruning metric. <u>Underscored</u> results indicate the default pruning configuration of each method.

The results are shown in Table 3.7. We refer the reader to Chapter 3.8.1 for analysis on image classifiers and Chapter 3.8 for analysis on previous LLMs. The default pruning configuration of Wanda delivers the best pruned model (perplexity 7.26). Interestingly, for the magnitude metric, comparing weights of the same input neuron (input, 1) yields a perplexity of 8.86, significantly better than other grouping options. Three methods also produce equivalent pruning results as under this comparison group – the input is the same, thus weight ranking only depends on weight magnitude. This finding further highlights the importance of using a proper comparison group for pruning LLMs, even for the classical magnitude pruning approach.

Robustness to Calibration Samples. We vary the number of calibration samples by selecting different sample sizes ranging between 1 and 256. Results are summarized in Figure 3.2. We see a clear difference in trend as the size of calibration data changes, where Wanda is much more robust when there are few calibration samples. Notably, even with a *single* sample, pruned

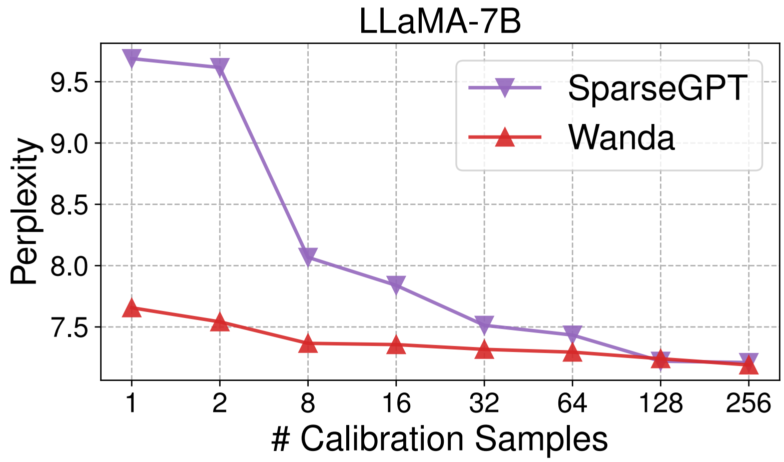


Figure 3.2: Wanda is more robust with less data.

networks obtained by Wanda have a perplexity of 7.66. This may be because input norm statistics $\|\mathbf{X}_j\|$ could be much easier to estimate than the full inverse hessian $\mathbf{H}^{-1}$ of the local layer-wise reconstruction problem.

Weight Update. We characterize the conditions under which the weight update process in SparseGPT can improve the effectiveness of pruning LLMs. We experiment with two ways of applying weight update: sequential and iterative. A sequential update means that at each layer, the full pruned mask is first computed and weight update is performed on the remaining weights. An iterative update means that the pruning and weight update steps proceed iteratively within each layer. SparseGPT adopts an iterative update procedure every 128 input channels, as it was found to give more accurate results.

Pruning Configuration		Weight Update	Sparsity		
Pruning Metric	Comparison Group		50%	4:8	2:4
Magnitude: $\|\mathbf{W}_{ij}\|$	layer	✗	17.59	16.84	42.13
	layer	Sequential	**12.56**	**13.37**	**21.36**
	(input, 128)	Iterative	26.77	36.98	47.61
Wanda : $\|\mathbf{W}_{ij}\| \cdot \|\mathbf{X}_j\|$	(output, 1)	✗	**7.26**	**8.57**	11.53
	(output, 1)	Sequential	7.32	8.59	**10.89**
	(input, 128)	Iterative	**7.26**	8.68	11.43

Table 3.8: Effects of the weight update. It offers little or negligible improvement to Wanda.

Effects of the weight update on magnitude pruning and Wanda are summarized in Table 3.8. We study these two pruning methods because they do not involve any weight update by default. An iterative update changes the comparison group for unstructured pruning, which we denote in the table as (input, 128). We make several interesting observations:

- For all considered sparsities, weight update can improve magnitude pruning by a large margin.
- For unstructured 50% and 4:8 sparsities, weight update does not bring any improvement to Wanda.
- For 2:4 sparsity, the improvement (from 11.53 to 10.89) is marginal. Note that the best 2:4 sparse model (10.89) we obtained here is better than that obtained by SparseGPT (11.00 in Table 3.3).

Last, we examine an extreme sparsity level (70%), where weight update can improve Wanda from 84.50 to 29.65. However, the best pruned model (29.65) lags far behind the dense LLaMA-7B (5.68).

3.7 Related Work

Network Pruning and Sparsity. Pruning is a popular technique for compressing neural networks through the elimination of weights, yielding sparse networks [77, 78]. It can be broadly categorized into *structured* and *unstructured* approaches.

Structured pruning methods [50, 51, 55, 93, 94, 125, 142], sometimes referred to as activation pruning [49, 67], remove entire structured components of a network, facilitating efficient GPU speedups. Some existing methods [46, 47] have explored structured pruning based on activation statistics of neuron/filter output, e.g. percentage of zero activations [48] and activation mean [119]. Recently, [112] have studied structured pruning of LLMs. [56, 91] and [60] have demonstrated the existence of prompt-dependent and task-specific sparsity in the structural components of LLMs, e.g., attention heads and MLP neurons.

Unstructured methods [66, 81, 116, 117, 118, 123] like magnitude pruning operate at the individual weight level, maintaining performance even at higher sparsity levels. Existing pruning methods usually require either modifications to the training procedure [68, 85], retraining the pruned networks to regain accuracy [54, 74], or an even more computationally intensive iterative retraining process [80, 82]. However, scaling these methods to LLMs with billions of parameters presents a challenge, as the required training process demands substantial computational resources [64, 106].

Pruning with Limited Data. Most related to our approach is a recent line of work on pruning with limited data [73, 75, 76, 83]. Such methods require no modification to the original training procedure and also no retraining of the pruned networks on the full training dataset. The primary aim of these methods is to preserve performance during the pruning procedure, assuming access to a limited and small amount of data, also referred to as the calibration data. In order to mitigate the accuracy drop, a layer-wise reconstruction problem [73] is solved to minimize the change of output evaluated on the calibration data. Existing solvers [75, 84] for the layer-wise reconstruction problem rely on heavy computation of second-order Hessian inverses, which do not scale to the large hidden state size of LLMs. SparseGPT [59] develops an efficient weight update procedure for LLMs via synchronized second-order Hessian updates.

Emergent Properties of LLMs. Our work is also related to recent studies on the existence of large magnitude outlier features in large language models [90, 120, 135, 136, 137, 138]. [58]

demonstrate that when LLMs exceed a certain parameter scale (e.g., 6B), large magnitude features start to emerge and strongly affect all layers, which can be seen as an emergent property of LLMs [58, 86, 89]. They also pinpoint these emerging features as the reason why existing quantization methods fail. This observation has spurred the development of various quantization schemes [57, 58, 147, 150, 155] tailored specifically for LLMs to handle outlier features. Our work extends this understanding, demonstrating that outlier features should also serve as pivotal indicators of which weights to prune in LLMs.

3.8 Additional Results

3.8.1 Image Classifiers

We study how Wanda would perform against magnitude pruning on tasks where the latter has been widely used. We conduct a study on ImageNet-1K [127], a standard image classification task where magnitude pruning has been extensively studied [67, 95]. We consider two modern vision architectures: ConvNeXt [128] and Vision Transformer (ViT) [130]. We choose these two architectures mainly for two reasons: first, as LLMs are based on Transformers, we would like to test if our observations on LLMs still hold on Transformers for other tasks; second, as we are evaluating on image classification, we are interested in examining how these pruning methods work on ConvNet models, with ConvNeXt being a representative architecture.

We use two ImageNet-1K pretrained models: ConvNeXt-B and DeiT-B, with a top-1 accuracy of 83.8% and 81.8% respectively. We prune the linear layers only (for ConvNeXt, this includes equivalent 1×1 convolution layers). For calibration data, we sample 4096 images from ImageNet training set. We observe that 4096 samples lead to a stable result for our pruning metric, beyond which we notice only a marginal effect. We report the accuracy of one-shot pruned models without any subsequent retraining.

We first study whether pruning *per output* is superior over pruning *per layer* for pruning image classifiers. In Figure 3.3, we show comparison results for both the magnitude metric and the pruning metric of Wanda. We can see that for both ConvNeXt-B and DeiT-B, layer-wise pruning is slightly better than pruning *per output*. We then compare the pruning metric of Wanda and the magnitude metric on layer-wise pruning. Results are shown in Figure 3.4. Our novel pruning metric leads to better results than magnitude pruning, especially at high sparsities (e.g., 70% and 80%).

3.8.2 Previous LLMs

In addition to LLaMA and LLaMA-2, we experiment with three previous LLM model families: namely OPT [64], BLOOM [65] and Pythia [141].

Comparison with Baselines. For OPT and Pythia, we experiment with varying sparsity levels (10% to 50%). We conduct additional evaluation on OPT and BLOOM models with various sizes. Results are shown in Table 3.9, Table 3.10 and Table 3.11 respectively. Our observations are as follows:

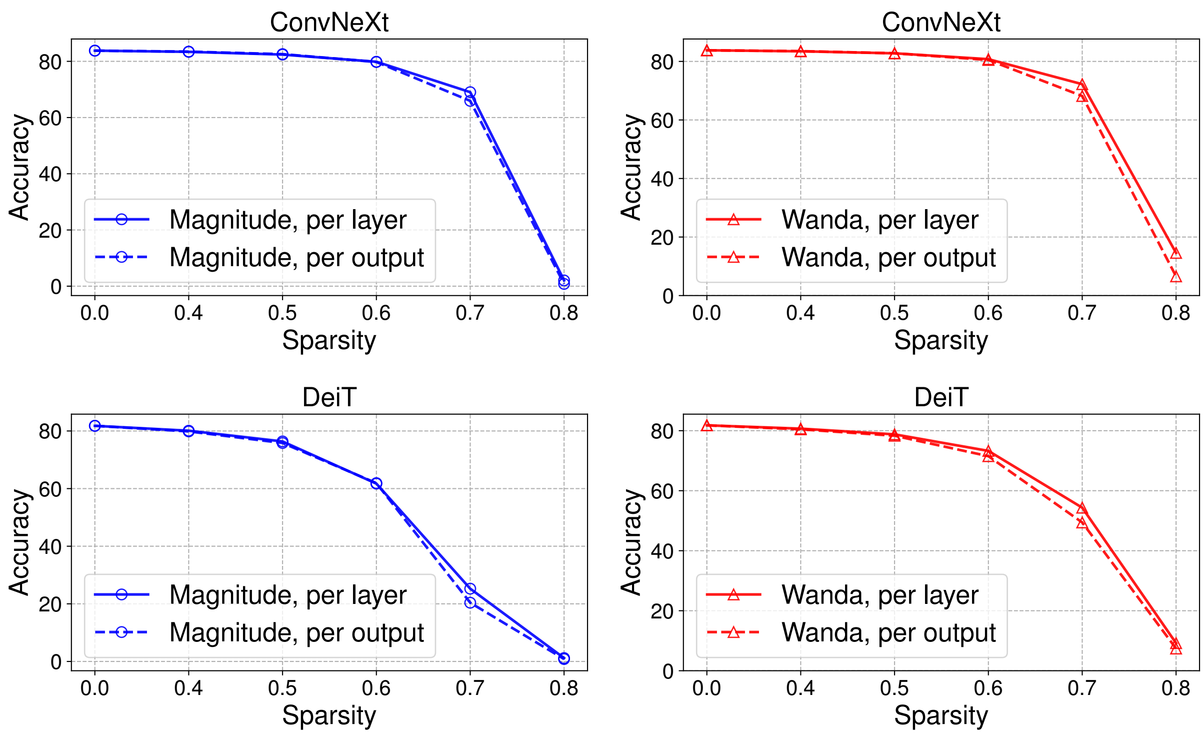


Figure 3.3: Analysis of comparison groups on pruning image classifiers.

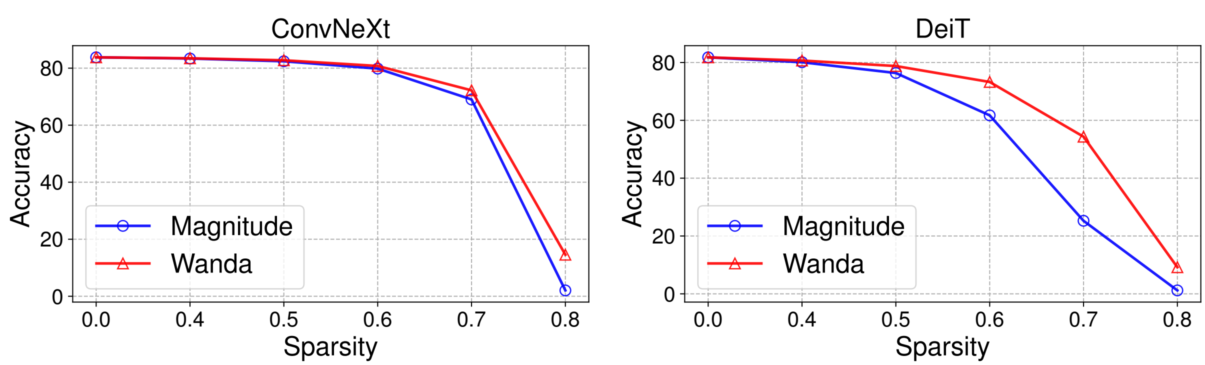


Figure 3.4: Our pruning metric outperforms the magnitude metric on pruning image classifiers.

- Unlike LLaMA and LLaMA-2, the well-established magnitude pruning approach fails catastropically on OPT-13B and Pythia-12B, even for low sparsity levels (e.g., 20%). This result further highlights the limitations of magnitude pruning for LLMs, as discussed in Section 3.4.
- Unlike magnitude pruning, Wanda successfully prunes these LLMs to much higher sparsities across various LLM model families, without any weight update on the kept weights. This result shows that LLMs have effective sub-networks that are *exact*. We hope this observation could contribute to a better understanding of sparsity in LLMs.
- There are cases where Wanda slightly underperforms SparseGPT, especially for OPT models (see Table 3.10), suggesting that for OPT, there may be a tradeoff between pruning speed

and pruning accuracy. However, the gap between SparseGPT and Wanda tends to get smaller as model sizes increase. This can be seen in Table 3.10 and Table 3.11.

- At lower sparsities (e.g., 20%), Table 3.9 indicates that the computationally intensive weight update process may be unnecessary, as Wanda yields comparable or slightly superior results.

				Sparsity				
Model	Dense	Pruning Method	Weight Update	10%	20%	30%	40%	50%
OPT-13B	10.13	Magnitude	✗	14.45	9e3	1e4	1e4	1e4
		SparseGPT	✓	10.11	10.10	10.12	**10.35**	**11.19**
		Wanda	✗	**10.09**	**10.07**	**10.09**	10.63	11.42
Pythia-12B	8.59	Magnitude	✗	127.76	2e5	7e5	2e5	3e5
		SparseGPT	✓	**8.59**	8.65	8.86	9.39	**11.02**
		Wanda	✗	**8.59**	**8.60**	**8.85**	**9.31**	11.27

Table 3.9: Pruning Pythia-13B and OPT-13B with various sparsity levels.

			OPT					
Method	Weight Update	Sparsity	125m	350m	1.3B	2.7B	6.7B	13B
Dense	-	50%	27.66	22.00	14.62	12.47	10.86	10.13
Magnitude	✗	50%	7e3	6e3	1e4	9e3	9e4	2e4
SparseGPT	✓	50%	**37.07**	**34.76**	**17.44**	**13.48**	**11.57**	**11.19**
Wanda	✗	50%	38.96	35.92	19.12	14.28	11.94	11.42

Table 3.10: Pruning OPT family models with various sizes.

			BLOOM				
Method	Weight Update	Sparsity	560m	1.1B	1.7B	3B	7.1B
Dense	-	50%	22.42	17.68	15.39	13.48	11.37
Magnitude	✗	50%	2e10	1e6	2e5	8e6	2e6
SparseGPT	✓	50%	**28.92**	**21.35**	**18.88**	16.76	13.96
Wanda	✗	50%	30.74	22.72	19.79	**16.45**	**13.55**

Table 3.11: Pruning BLOOM family models with various sizes.

Comparison Group We test if our observation regarding pruning *per output* holds true for other LLM model families. We experiment on OPT [64] and BLOOM [65]. In Table 3.12 and Table 3.13, we provide results comparing pruning *per layer* and pruning *per output* for these two LLM model families. The pruning metric is fixed to be our proposed metric: $|\mathbf{W}_{ij}| \cdot \|\mathbf{X}_j\|$. We can see that our findings regarding the comparison group are not limited to LLaMA. For OPT and BLOOM model families, pruning *per output* consistently outperforms pruning *per layer*.

		OPT					
Comparison Group	Sparsity	125m	350m	1.3B	2.7B	6.7B	13B
per layer	50%	46.95	38.97	22.20	22.66	15.35	13.54
per output	50%	**38.96**	**36.19**	**19.42**	**14.22**	**11.97**	**11.42**

Table 3.12: Comparison of pruning *per layer* versus *per output* for OPT models.

		BLOOM				
Comparison Group	Sparsity	560m	1.1B	1.7B	3B	7.1B
per layer	50%	34.57	26.26	22.55	18.22	15.31
per output	50%	**30.74**	**22.72**	**19.79**	**16.45**	**13.55**

Table 3.13: Comparison of pruning *per layer* versus *per output* for BLOOM models.

3.8.3 Additional Baselines

We compare with several prior activation pruning methods. These approaches remove entire neurons in the network based on certain statistics of the neuron output: mean and standard deviation [119], correlation [47] and mean squared norm [46]. We show the results of pruning LLaMA-7B in Table 3.14. We compute these output statistics using the calibration set and remove neurons with smaller values. We observe that these activation pruning methods are unable to prune LLMs effectively.

We also compare with several prior methods on pruning BERT [18]: SNIP [129], BERT-LTH [144], Movement [68], Platon [131] and PINS [92]. In Table 3.15, we provide a summary of existing pruning methods, mostly for pruning BERT. A key distinction of these methods and our work is that they interleave pruning heavily with the fine-tuning process. Another difference is that BERT pruning methods focus on performance on a downstream task, rather than preserving the general performance of pretrained language models.

We adopt these prior methods for pruning LLMs, where the goal is to preserve the language modeling ability. Thus we use the pre-training auto-regressive loss to compute their pruning metrics. We evaluate two settings: one-shot pruning and one-shot pruning followed by fine-tuning. For one-shot pruning, we use the pruning metrics listed in Table 3.15 to prune LLMs. We fine-tune the pruned LLMs within a limited computational budget, i.e., one day. Results are summarized in Table 3.16. We observe that these pruning methods are not effective when adapted for pruning LLMs.

Model	Dense	Activation Statistics	Sparsity 10%	20%	30%	40%	50%
LLaMA-7B	5.68	Mean	1e5	2e5	3e5	3e5	3e5
		Standard Deviation	161	649	7e3	1e5	2e5
		Correlation	1e4	7e4	2e5	2e5	2e5
		Mean Squared Norm	16.43	98.13	9e2	1e5	4e5

Table 3.14: Results for activation pruning methods.

Pruning Method	Pruning Type	Pruning Metric	Training Procedure
SNIP	Unstructured	Loss Sensitivity	Pruning at Initialization
BERT-LTH	Unstructured	Magnitude	Fine-tuning BERT
Movement	Unstructured	Loss Sensitivity	Fine-tuning BERT
Platon	Unstructured	Loss Sensitivity	Fine-tuning BERT
PINS	Unstructured	Loss Sensitivity	Fine-tuning BERT

Table 3.15: Summary of prior pruning methods on BERT.

Model	Dense	Fine-tuning	Pruning method SNIP	BERT-LTH	Movement	Platon	PINS	Wanda
LLaMA-7B	5.68	✗	231.48	17.29	349.33	124.91	89.12	**7.26**
		✓	102.32	12.43	168.17	102.34	72.13	**6.28**

Table 3.16: Comparisons with prior pruning methods on BERT (unstructured 50% sparsity).

3.8.4 Number of Calibration Samples

In the main paper, the default number of calibration samples is 128. This choice is adopted from [59], which was selected on the OPT model family [64]. Here we conduct a detailed analysis on the effect of the number of calibration samples for LLaMA and LLaMA-2 model families. We show the results for pruning LLaMA-7B and LLaMA-2-7B with unstructured 50% sparsity in Table 3.17. We find that there is a slight improvement in performance of pruned LLMs when the size of calibration set goes beyond 128.

Model	Method	1	16	32	64	128	256	512	1024	2048
LLaMA-7B	SparseGPT	10.22	7.61	7.36	7.29	7.26	7.20	7.19	7.23	7.20
	Wanda	7.46	7.27	7.28	7.28	7.26	7.30	7.26	7.25	7.26
LLaMA-2-7B	SparseGPT	8.63	6.67	6.62	6.61	6.53	6.52	6.50	6.49	6.49
	Wanda	6.53	6.45	6.46	6.45	6.45	6.45	6.45	6.45	6.45

Table 3.17: WikiText validation perplexity of pruned LLaMA and LLaMA-2 under various number of calibration samples, with 50% sparsity.

3.8.5 Robustness Analysis

In this part, we perform a robustness analysis of our results in Section 3.5.2. The result in Table 3.3 is evaluated under a fixed calibration set. Since both SparseGPT and Wanda require calibration data to estimate input statistics, we sample different calibration sets under 5 random seeds and evaluate these two pruning methods. In Table 3.18, we report the perplexity (mean and standard deviation) of pruned LLaMA models under 5 random seeds. In many cases, the variance across random seeds is lower for Wanda, suggesting that Wanda is more stable with variations in the calibration sets.

			LLaMA		LLaMA-2	
Method	Weight Update	Sparsity	7B	13B	7B	13B
Dense	-	0%	5.68	5.09	5.12	4.57
Magnitude	✗	50%	17.29	20.21	14.89	6.37
SparseGPT	✓	50%	**7.25** (±0.03)	6.24 (±0.02)	6.52 (±0.02)	5.63 (±0.01)
Wanda	✗	50%	**7.25** (±0.01)	**6.18** (±0.01)	**6.44** (±0.01)	**5.59** (±0.01)
Magnitude	✗	4:8	16.84	13.84	16.48	6.76
SparseGPT	✓	4:8	8.67 (±0.08)	**7.43** (±0.03)	8.05 (±0.03)	6.59 (±0.04)
Wanda	✗	4:8	**8.65** (±0.01)	**7.43** (±0.03)	**7.98** (±0.01)	**6.56** (±0.01)
Magnitude	✗	2:4	42.13	18.37	54.59	8.33
SparseGPT	✓	2:4	**10.94** (±0.23)	**9.08** (±0.04)	**10.44** (±0.42)	**8.28** (±0.05)
Wanda	✗	2:4	11.48 (±0.05)	9.60 (±0.04)	11.10 (±0.09)	**8.28** (±0.02)

Table 3.18: WikiText validation perplexity of pruned LLaMA and LLaMA-2 models. We report the mean and standard deviation under 5 random seeds.

3.8.6 Higher Sparsity

In Section 3.5, we have evaluated unstructured pruning with a sparsity level of 50%. This is to follow the evaluation setup of [59]. In this part, we evaluate on higher sparsity levels, i.e., 60%

and 80%. Results for these two sparsity levels are shown Table 3.19 and Table 3.20 respectively. At 60% sparsity, Wanda remains competitive with SparseGPT. At 80% sparsity, SparseGPT is able to outperform Wanda, but the performance drop compared to the dense counterpart is significant. The best 80% sparse model (25.86) underperforms the smallest dense LLaMA-7B model (5.68) by a large gap. This suggests that at extreme sparsity levels, it may be better to use a small dense model trained to convergence instead.

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
Magnitude	✗	60%	6e2	2e2	27.67	9.34	4e3	11.23	8.21
SparseGPT	✓	60%	**10.51**	**8.56**	6.66	**5.82**	**9.58**	7.80	**4.98**
Wanda	✗	60%	10.66	**8.56**	**6.49**	5.83	9.71	**7.75**	**4.98**

Table 3.19: WikiText validation perplexity of pruned LLaMA and LLaMA-2 models with unstructured 60% sparsity.

			LLaMA				LLaMA-2		
Method	Weight Update	Sparsity	7B	13B	30B	65B	7B	13B	70B
Dense	-	0%	5.68	5.09	4.77	3.56	5.12	4.57	3.12
Magnitude	✗	80%	1e5	3e4	1e5	2e4	nan	5e4	3e4
SparseGPT	✓	80%	**2e2**	**1e2**	**54.98**	**32.80**	**1e2**	**1e2**	**25.86**
Wanda	✗	80%	5e3	4e3	2e3	2e3	5e3	2e3	1e2

Table 3.20: WikiText validation perplexity of pruned LLaMA and LLaMA-2 models with unstructured 80% sparsity.

3.8.7 Few-shot Results on MMLU

Our experiments in Section 3.5.1 focus on zero-shot evaluation. However, LLMs are also known for their ability to learn in context. In this part, we conduct additional evaluation on few-shot tasks. Specifically, we choose the Massive Multitask Language Understanding benchmark (MMLU) [88]. In alignment with the evaluation methodology of [22], we perform 5-shot evaluation. In Table 3.21, we report the mean accuracies for both dense LLMs and sparse LLMs with unstructured 50% sparsity. In the few-shot setting, Wanda performs competitively with SparseGPT. Notably, large sparse LLMs surpass smaller dense counterparts, e.g., sparse LLaMA-13B/LLaMA-2-13B versus dense LLaMA-7B/LLaMA-2-7B. This trend can not be observed from the standard magnitude pruning approach.

			LLaMA		LLaMA-2	
Method	Weight Update	Sparsity	7B	13B	7B	13B
Dense	-	0%	39.85	52.92	52.08	61.69
Magnitude	✗	50%	30.69	30.69	32.14	48.76
SparseGPT	✓	50%	**34.43**	45.08	38.68	54.83
Wanda	✗	50%	33.49	**46.04**	**39.27**	**55.01**

Table 3.21: 5-shot results (mean accuracies %) on MMLU for unstructured 50% sparsity.

3.8.8 Fine-tuning

In Table 3.6 of Section 3.6, we report the mean zero-shot accuracies after fine-tuning Wanda pruned LLaMA-7B models. In this part, we report the task-wise performance of these fine-tuned models. Results are summarized in Table 3.22. For per-task accuracies, most of the performance drop during pruning can be recovered through fine-tuning. Note that here we are performing limited fine-tuning with a computational budget (12 hours for LoRA fine-tuning and 3 days for full parameter fine-tuning). It remains to be seen if the gap between sparse pruned LLMs and the dense counterparts can be fully recovered given more computational budget.

Sparsity	Fine-tuning	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
Dense	-	75.05	66.43	56.92	69.93	75.34	41.89	34.40	59.99
50%	✗	71.22	55.60	51.85	66.06	69.11	36.86	28.80	54.21
	LoRA	72.90	60.79	55.36	67.48	71.42	37.97	29.80	56.53
	Full	**74.50**	**62.84**	**55.83**	**69.02**	**73.49**	**39.20**	**32.20**	**58.15**
4:8	✗	70.97	58.24	46.81	65.83	65.53	33.97	28.00	52.76
	LoRA	71.24	60.04	54.47	66.14	67.68	35.32	29.20	54.87
	Full	**73.32**	**60.99**	**55.21**	**66.80**	**71.76**	**36.46**	**32.00**	**56.65**
2:4	✗	69.30	51.99	42.06	62.75	60.94	28.07	24.60	48.53
	LoRA	70.32	**64.98**	52.53	65.04	67.00	33.53	27.80	54.46
	Full	**73.21**	61.34	**54.86**	**66.18**	**70.24**	**35.68**	**31.80**	**56.19**

Table 3.22: The gap between pruned LLMs and dense LLMs can be largely mitigated via fine-tuning.

3.8.9 Zero-Shot Tasks

For zero-shot results in Section 3.5.1, the 7 evaluated zero-shot tasks are: BoolQ [98], RTE [99], HellaSwag [100], WinoGrande [101], ARC Easy and Challenge [102], and OpenbookQA [103]. For reproducibility, we used commit `df3da98` on the main branch. All tasks were evaluated on task version 0 except for BoolQ, where the evaluated version was 1. We show the task-wise performance in Table 3.23,3.24,3.25,3.26,3.27 and 3.28.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	75.05	66.43	56.92	69.93	75.34	41.89	34.40	59.99
	Magnitude	54.59	54.51	45.49	59.19	58.84	33.53	22.40	46.94
	SparseGPT	**72.05**	54.15	51.43	**67.88**	**71.38**	**37.71**	**30.00**	**54.94**
	Wanda	71.22	**55.60**	**51.85**	66.06	69.11	36.86	28.80	54.21
13B	Dense	77.89	70.4	59.94	72.77	77.40	46.50	33.20	62.59
	Magnitude	54.89	51.26	44.16	63.14	58.80	33.79	27.20	47.61
	SparseGPT	**76.97**	61.01	54.95	71.67	72.47	41.98	31.20	58.61
	Wanda	75.90	**62.82**	**55.71**	**71.98**	**73.19**	**43.52**	**32.20**	**59.33**
30B	Dense	82.69	66.79	63.35	75.69	80.30	52.82	36.00	65.38
	Magnitude	64.34	50.18	50.59	66.54	72.39	43.77	29.00	53.83
	SparseGPT	**82.32**	62.45	59.15	**75.22**	78.96	48.56	**35.00**	63.09
	Wanda	81.90	**65.34**	**60.93**	73.48	**79.29**	**49.66**	34.60	**63.60**
65B	Dense	84.83	69.68	64.54	77.27	81.40	52.90	38.20	66.97
	Magnitude	79.15	62.45	61.90	74.74	76.40	49.57	35.00	62.74
	SparseGPT	84.60	70.76	63.90	**77.43**	79.35	**50.85**	37.20	66.30
	Wanda	**84.70**	**71.48**	**64.55**	76.87	**79.75**	50.51	**38.80**	**66.67**

Table 3.23: Accuracies (%) of LLaMA for 7 zero-shot tasks with unstructured 50% sparsity.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	75.05	66.43	56.92	69.93	75.34	41.89	34.40	59.99
	Magnitude	51.19	50.54	46.73	60.69	58.96	30.89	23.20	46.03
	SparseGPT	**73.06**	58.12	**47.88**	**65.98**	**66.75**	32.42	25.40	**52.80**
	Wanda	70.97	**58.24**	46.81	65.83	65.53	**33.97**	**28.00**	52.76
13B	Dense	77.89	70.40	59.94	72.77	77.40	46.50	33.20	62.59
	Magnitude	61.07	51.26	48.91	65.11	63.26	35.67	28.40	50.53
	SparseGPT	**76.61**	57.76	51.24	70.17	**71.17**	37.20	27.80	55.99
	Wanda	74.89	**57.89**	**51.26**	**70.56**	70.29	**37.97**	**29.80**	**56.09**
30B	Dense	82.69	66.79	63.35	75.69	80.30	52.82	36.00	65.38
	Magnitude	63.55	50.18	49.45	65.75	73.36	42.83	29.60	53.53
	SparseGPT	**78.69**	**61.73**	56.15	**74.35**	76.94	46.08	31.60	60.79
	Wanda	77.38	58.80	**58.79**	74.28	**77.34**	**46.46**	**34.00**	**61.00**
65B	Dense	84.83	69.68	64.54	77.27	81.40	52.90	38.20	66.97
	Magnitude	74.95	68.23	60.85	74.27	76.45	47.61	32.80	62.17
	SparseGPT	**84.35**	68.95	**61.00**	**77.19**	78.75	48.46	35.40	64.87
	Wanda	84.29	**70.92**	59.54	76.64	**79.00**	**48.83**	**35.60**	**64.97**

Table 3.24: Accuracies (%) of LLaMA for 7 zero-shot tasks with 4:8 sparsity.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	75.05	66.43	56.92	69.93	75.34	41.89	34.40	59.99
	Magnitude	53.09	55.60	42.30	59.91	53.28	27.13	21.80	44.73
	SparseGPT	**70.46**	**60.65**	**42.99**	**64.88**	**61.49**	**30.12**	23.60	**50.60**
	Wanda	69.30	51.99	42.06	62.75	60.94	28.07	**24.60**	48.53
13B	Dense	77.89	70.40	59.94	72.77	77.40	46.50	33.20	62.59
	Magnitude	60.95	49.10	45.81	62.75	58.75	31.06	27.60	48.00
	SparseGPT	**72.14**	**55.23**	**48.11**	**68.98**	**66.71**	**34.98**	26.40	**53.22**
	Wanda	70.21	53.43	46.74	68.82	65.82	33.87	**27.20**	52.30
30B	Dense	82.69	66.79	63.35	75.69	80.30	52.82	36.00	65.38
	Magnitude	65.11	52.35	51.72	66.22	70.88	38.23	27.60	53.16
	SparseGPT	**75.60**	62.13	53.10	72.61	**75.13**	41.98	31.80	58.91
	Wanda	74.68	**63.80**	**54.41**	**72.93**	74.41	**42.06**	**32.20**	**59.21**
65B	Dense	84.83	69.68	64.54	77.27	81.40	52.90	38.20	66.97
	Magnitude	77.9	64.98	58.65	72.85	75.15	45.05	34.40	61.28
	SparseGPT	83.15	65.34	**57.20**	**76.72**	78.20	45.18	32.20	62.57
	Wanda	**83.58**	**66.79**	56.36	75.82	**78.23**	**45.56**	**33.60**	**62.84**

Table 3.25: Accuracies (%) of LLaMA for 7 zero-shot tasks with 2:4 sparsity.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	77.74	62.82	57.17	68.90	76.39	43.52	31.40	59.71
	Magnitude	63.00	**57.04**	49.13	63.30	64.10	34.64	26.80	51.14
	SparseGPT	75.02	54.15	52.37	**69.85**	**73.27**	**39.85**	29.20	**56.24**
	Wanda	**75.99**	53.43	**52.49**	68.19	72.77	39.59	**31.20**	**56.24**
13B	Dense	80.52	65.34	60.06	72.22	79.42	48.46	35.20	63.03
	Magnitude	57.61	55.96	54.40	65.27	70.54	38.40	27.80	52.85
	SparseGPT	81.44	**65.34**	55.83	**72.77**	74.83	42.24	32.60	60.72
	Wanda	**81.84**	64.02	**56.90**	71.35	**76.18**	**43.52**	**32.00**	**60.83**
70B	Dense	83.40	67.87	66.10	78.06	82.55	54.44	37.20	67.08
	Magnitude	70.55	60.65	61.50	73.48	75.70	49.23	35.40	60.93
	SparseGPT	**83.55**	70.40	63.80	**78.85**	**82.40**	**53.75**	**38.20**	**67.28**
	Wanda	82.50	**73.65**	**64.10**	78.14	80.80	52.65	37.40	67.03

Table 3.26: Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with unstructured 50% sparsity.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	77.74	62.82	57.17	68.90	76.39	43.52	31.40	59.71
	Magnitude	63.00	52.35	50.08	62.43	64.73	35.92	26.00	50.64
	SparseGPT	72.69	**55.23**	**48.20**	**68.11**	**69.15**	**35.84**	**27.40**	**53.80**
	Wanda	**73.91**	53.79	46.45	66.61	66.71	34.13	25.80	52.49
13B	Dense	80.52	65.34	60.06	72.22	79.42	48.46	35.20	63.03
	Magnitude	63.33	57.76	53.96	64.40	68.48	35.75	26.00	52.81
	SparseGPT	79.97	**66.79**	52.01	**70.64**	73.61	41.04	**30.00**	**59.15**
	Wanda	**80.26**	65.62	**52.05**	69.48	**73.88**	**41.54**	28.40	58.75
70B	Dense	83.40	67.87	66.10	78.06	82.55	54.44	37.20	67.08
	Magnitude	70.95	59.21	60.05	74.11	76.25	46.76	34.60	60.28
	SparseGPT	82.20	**72.20**	61.45	**77.82**	**80.85**	51.19	35.20	65.84
	Wanda	**84.30**	71.80	**61.90**	76.24	80.40	**51.80**	**36.00**	**66.06**

Table 3.27: Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with 4:8 sparsity.

Params	Method	BoolQ	RTE	HellaSwag	WinoGrande	ARC-e	ARC-c	OBQA	Mean
7B	Dense	77.74	62.82	57.17	68.90	76.39	43.52	31.40	59.71
	Magnitude	56.23	51.35	42.27	60.93	59.18	27.31	21.80	45.58
	SparseGPT	**70.52**	**58.84**	**43.26**	**66.69**	**64.10**	29.97	23.20	**50.94**
	Wanda	67.65	53.07	40.92	62.43	61.78	**31.20**	**24.20**	48.75
13B	Dense	80.52	65.34	60.06	72.22	79.42	48.46	35.20	63.03
	Magnitude	65.69	54.15	50.13	62.04	62.46	31.74	23.00	49.89
	SparseGPT	76.79	59.38	46.58	**68.67**	**70.62**	36.60	25.40	54.86
	Wanda	**76.80**	**61.22**	**47.82**	66.90	69.24	**36.82**	**26.40**	**55.03**
70B	Dense	83.40	67.87	66.10	78.06	82.55	54.44	37.20	67.08
	Magnitude	73.20	57.04	58.40	74.27	76.15	45.22	35.40	59.95
	SparseGPT	79.50	**70.76**	59.00	**76.64**	78.95	**48.55**	33.80	63.89
	Wanda	**82.20**	69.85	**59.34**	76.23	**79.30**	47.26	**34.80**	**64.14**

Table 3.28: Accuracies (%) of LLaMA-2 for 7 zero-shot tasks with 2:4 sparsity.

3.9 Conclusion

In this work, we propose a simple and effective method for pruning Large Language Models (LLMs). Inspired by the recent discovery of emergent large magnitude features in LLMs, our approach, termed Wanda (Pruning by **W**eights **and a**ctivations), removes weights with the smallest magnitudes multiplied by the corresponding input activation norms, on a *per-output* basis. Without the need for any retraining or weight update procedures, Wanda is able to identify effective sparse networks within pretrained LLMs. We hope our work contributes to a better understanding of sparsity in LLMs. Last, considering the fast speed of pruning with Wanda, it would be interesting to investigate whether Wanda can be useful in the setting of sparse training [148, 149, 151, 152, 153], where pruning has to be conducted repeatedly and thus the pruning efficiency is critical.

Chapter 4

Massive Activations in Large Language Models

4.1 Overview

We observe an empirical phenomenon in Large Language Models (LLMs)—very few activations exhibit significantly larger values than others (e.g., 100,000 times larger). We call them *massive activations*. First, we demonstrate the widespread existence of massive activations across various LLMs and characterize their locations. Second, we find their values largely stay constant regardless of the input, and they function as indispensable bias terms in LLMs. Third, these massive activations lead to the concentration of attention probabilities to their corresponding tokens, and further, implicit bias terms in the self-attention output. Last, we also study massive activations in Vision Transformers. Code is available at `https://github.com/locuslab/massive-activations`.

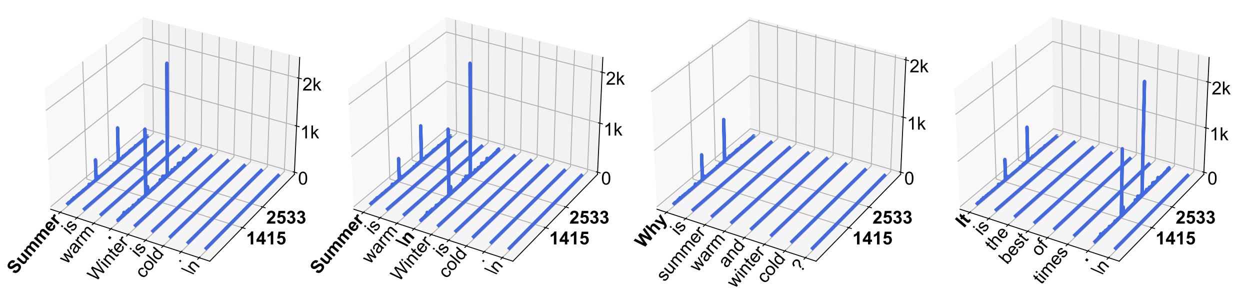


Figure 4.1: **Activation Magnitudes (z-axis) in LLaMA2-7B.** $\mathbf{x}$ and $\mathbf{y}$ axes are sequence and feature dimensions. For this specific model, we observe that activations with massive magnitudes appear in two fixed feature dimensions (1415, 2533), and two types of tokens—the starting token, and the first period (`.`) or newline token (`\n`).

4.2 Introduction

Large Language Models (LLMs) [3, 4] have demonstrated remarkable capabilities. The majority of existing studies conducted on these models are focused on their external behaviors, e.g., evaluating their performance on various tasks [105, 107], developing prompts to elicit accurate responses [9, 169]. While these studies are encouraging and highlight the potential of these models, it is also important to gain insights into their internal mechanisms, especially as they are being increasingly integrated into many real-world applications. However, research on the internal workings of these models remains relatively limited.

In this work, we discover and study a surprising phenomenon in the internal representations of LLMs. Examining the hidden states in these models, we find that certain activations exhibit huge magnitudes, e.g., more than 4 orders of magnitude larger than the median, and could take on absolute values larger than 15,000 in LLaMA2-70B [23], despite the presence of normalization layers. These activations are also extremely rare, often numbering fewer than 10 among tens of millions of total activations. Figure 4.1 illustrates this phenomenon in LLaMA2-7B. As these activations are so much larger in magnitudes compared to others, we name them *massive activations*. We demonstrate their presence in a wide range of LLMs, spanning different model sizes and families.

We explore where massive activations are located in LLMs. Regarding the depth dimension of LLMs, the appearance of massive activations is mostly abrupt: they emerge suddenly after a single layer of computation, and diminish at the last few layers. Further, we find massive activations occur in a small number of feature dimensions that are input agnostic. Many of these activations are found within the starting word token and delimiter tokens. Additionally, we show that massive activations are not the same as outlier features [58], a previously known phenomenon in LLMs.

We show that massive activations act as fixed but crucial bias terms in LLMs. Here by bias terms, we mean certain internal states of the models that are independent from the inputs, analogous to the bias term b in a linear layer $y = Wx + b$. First, we show that massive activations play a critical role in LLMs' capabilities. For instance, in LLaMA2-7B, setting merely four massive activations (out of millions of activations) to zero would result in catastrophic collapse in model performance. Further, setting them to their mean values does not hurt the model, suggesting their role is equivalent to simple constant biases. Our analysis reveals that after the initial layers, LLMs repurpose the tokens linked with massive activations to store these important biases.

Intriguingly, massive activations are closely connected with self-attention. In particular, we show massive activations cause attention to be attracted to the tokens associated with them. Our findings extend the observations from "attention sinks" [190]—we demonstrate that LLMs allocate excessive attention to more than just the first token, and provide an in-depth analysis on how such attention concentration patterns arise. Our analysis suggests that LLMs try to learn implicit bias components in self-attention via massive activations, during their pretraining phase. We thus experiment with augmenting self-attention with additional key and value embeddings that are explicitly designed as biases. Remarkably, we demonstrate that training with them eliminates the need for LLMs to learn massive activations.

Finally, we also observe massive activations in Vision Transformers (ViTs). They appear less frequently than those in LLMs but are still in many of the ViTs we have examined. In these ViTs, they tend to appear at fixed feature dimensions, but notably at varying patch tokens. Moreover, we find that these activations act similarly as fixed biases. Notably, we discuss the connections between massive activations and the recently proposed "register tokens" in ViTs [191]. We show they both learn values independent of input images, functioning as fixed biases. This offers an alternative interpretation for register tokens than that in the original work [191], where they were hypothesized to aggregate global image information.

4.3 Massive Activations: What are They?

We study autoregressive Transformers, which are built by a stack of L decoding layers. Each layer ℓ takes the previous hidden state $\mathbf{h}_{\ell-1} \in \mathbb{R}^{T\times d}$ as input and outputs a hidden state $h_\ell \in \mathbb{R}^{T\times d}$. T is the number of tokens and d is the number of features. Transformer layers use residual connections [30], and the computation can be formulated as:

$$h_\ell = h_{\ell-1} + \mathcal{F}_\ell(h_{\ell-1}) \tag{4.1}$$

where $\mathcal{F}_\ell$ is the residual transformation. Note that this includes both attention and MLP blocks. An *activation* denotes a specific scalar value in a hidden state. Unless otherwise specified, our study of activations is on the hidden state h_ℓ, i.e., the output of residual summations, not any intermediate states inside $\mathcal{F}_\ell$.

Existence in LLMs. We start with an illustrative example on LLaMA2-7B. In Figure 4.1, we visualize the intermediate features $\mathbf{h}_\ell$ of interest. We feed this model with short sentences and visualize the activation magnitudes ($\mathbf{z}$-axis) of the hidden states at a middle layer. $\mathbf{x}$ and $\mathbf{y}$ axes correspond to sequence and feature dimensions respectively. Each blue row corresponds to the feature embedding of one token. We observe up to four activations with significantly large magnitudes. The largest activation (about 2,000) is approximately 10,000 times larger than the median magnitude (about 0.2). The sheer scale of these activations makes them stand out from others. We thus refer to these special activations as *massive activations*.

Massive activations are not unique to this specific model LLaMA2-7B, but are widely observed in LLMs. In Figure 4.2 and Figure 4.3, we demonstrate the existence of massive activations in both LLaMA2-13B and Mixtral-8x7B [172]. Notably for Mixtral-8x7B, the largest activation magnitude can reach an absolute value of 7,000, around 4 orders of magnitude larger than the median feature magnitude (around 0.3). We refer the reader to Chapter 4.8 for results on more pretrained and fine-tuned LLMs.

Properties. We summarize two main properties of massive activations. The most notable property is that these activations possess massive values and their magnitudes are significantly larger than other activations, often several orders of magnitude larger than the median value. Another property is that they are exceptionally few in number. For LLaMA2-7B in Figure 4.1, there are approximately 40,000 total activations in each presented hidden state but at most four massive activations can be identified.

Quantitatively, we present the values of the top activation magnitudes in Table 4.1. We also provide a loose but broad definition: an activation qualifies as a massive activation if its magnitude

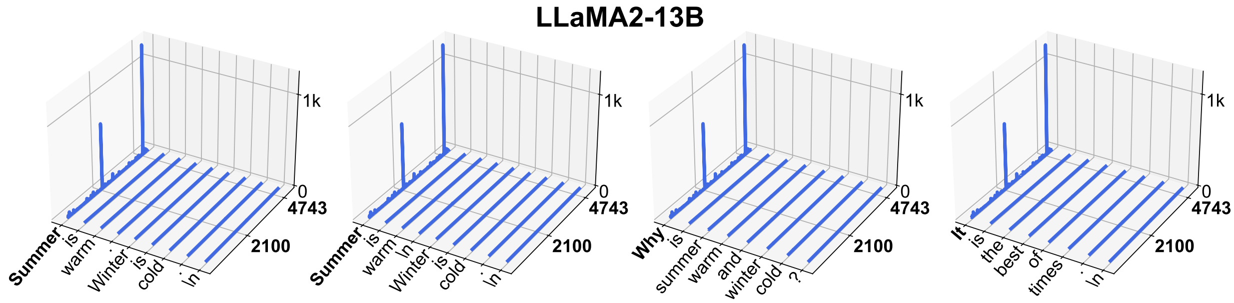


Figure 4.2: **Massive activations in LLaMA2-13B.** In this model, they appear in two fixed feature dimensions (2100, 4743), and are limited to the starting token.

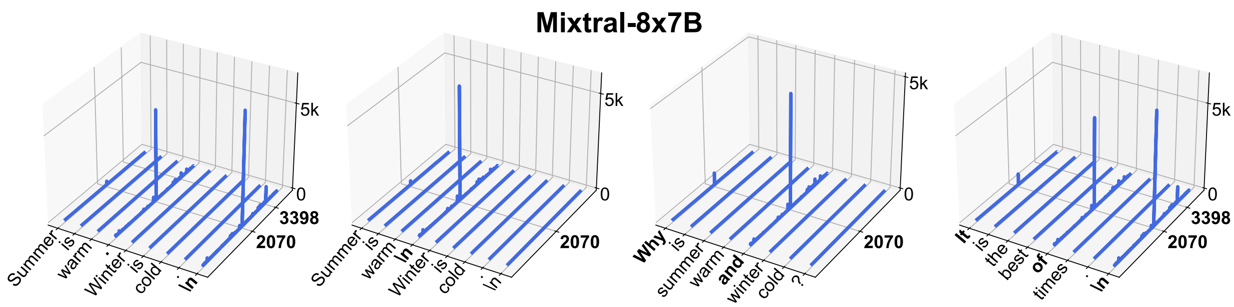


Figure 4.3: **Massive activations in Mixtral-8x7B.** In this model, they lie in two feature dimensions (2070, 3398), and are found within the starting token, delimiter tokens and certain word tokens ("`and`" and "`of`").

surpasses 100 and is at least or around 1,000 times larger than the median magnitude of its hidden state. We find this criterion to effectively identify these activations of interest across various LLMs, which are emphasized in bold in Table 4.1.

Next, we identify the locations of massive activations within LLMs. For a comprehensive analysis, rather than using short sentences as inputs, we collect 100 sequences (each with 4,096 tokens) from RedPajama [45]. We run LLMs on these 100 sequences and collect the hidden states from each layer.

Model	Top 1	Top 2	Top 3	Top 4	Top 5	Top-10	Top-100	Top 1%	Top 10%	median
LLaMA2-7B	**2622.0**	**1547.0**	**802.0**	**477.3**	156.9	45.7	10.6	1.1	0.6	0.2
LLaMA2-13B	**1264.0**	**781.0**	51.0	50.5	47.1	43.5	16.6	1.9	1.1	0.4
Mixtral-8x7B	**7100.0**	**5296.0**	**1014.5**	**467.8**	**302.8**	182.8	90.8	3.0	1.0	0.3

Table 4.1: Five largest, top 1% and 10%, and the median *activation magnitudes* at a hidden state of three LLMs. The activations that are considered as massive activations are highlighted in bold.

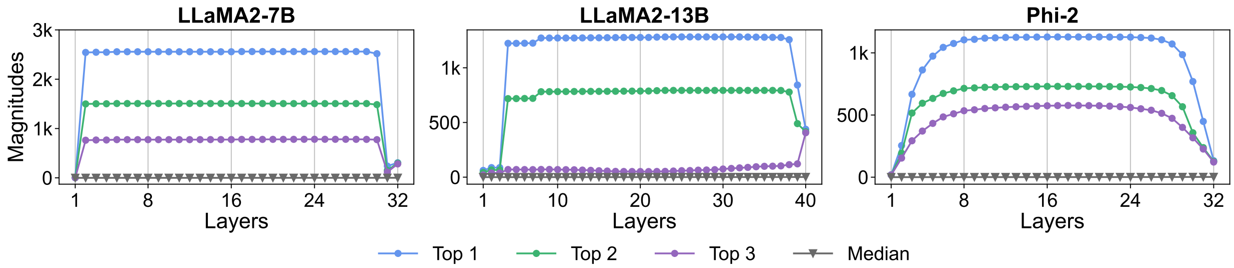


Figure 4.4: Three largest activation magnitudes and the median magnitude at each layer in LLMs.

4.3.1 Which Layers?

We determine the layers whose output hidden states exhibit massive activations. In Figure 4.4, we visualize the three largest activation magnitudes and the median of the hidden state output of each layer, with results averaged over 100 sequences. We examine three models: LLaMA2-7B, 13B and Phi-2 [161] (see Chapter 4.8.4 for more LLMs). In all cases, each of the top three activations comes from the same position in the hidden state across most of the middle layers. Generally, we observe the following:

Massive activations exist and remain as largely constant values throughout most of the intermediate layers. They emerge in the initial layers and start to diminish in the last few layers.

In LLaMA2-7B, massive activations first appear in layer 2 and remain nearly constant values until layer 30. Intriguingly, for LLaMA2-7B and 13B, massive activations emerge very rapidly from one layer of computation, e.g., layer 2 and layer 4 respectively. This means that they do not emerge as a result of gradual accumulation through many layers, and are caused by a rather different mechanism.

4.3.2 Which Feature and Sequence Dimensions?

We determine the locations of massive activations within hidden states, i.e., their feature and sequence dimensions. Since we have shown that their values largely stay constant in middle layers, we take on any such layer for this analysis.

LLaMA2-7B. In this model, massive activations are identified in two feature dimensions (1415 and 2533). Regarding sequence dimensions, we find that *massive activations appear at: 1. the starting word token, 2. the token representing the first period (`.`) or newline token (`\n`) in the sequence*. Figure 4.1 illustrates these findings for LLaMA2-7B. This is also consistent on long sequences. In cases where the input contains a "`.`" or "`\n`" token, four massive activations are observed. For the less common scenario where neither "`.`" nor "`\n`" is present, we can see two massive activations, both of which are associated with the initial token.

LLaMA2-13B. We find that massive activations in this model consistently appear in two feature dimensions, 2100 and 4743. *These activations are exclusively located within the starting token of the sequence, regardless of its semantics.* Figure 4.2 illustrates these behaviors within LLaMA2-

13B. For *any* given input sequence, only two massive activations are present, corresponding to features 2100 and 4743 of the first word token.

Mixtral-8x7B. For this model, massive activations lie in two feature dimensions, i.e., 2070 and 3398. For sequence dimensions, we find that *they are associated with the starting token, delimiter tokens and also certain word tokens, e.g., token "`and`" and token "`of`".* These word tokens tend to be conjunctions and prepositions, representing relatively few semantics. Figure 4.3 showcases these patterns in Mixtral-8x7B. Generally, for inputs of 4096 tokens in length, these tokens are predominantly located in the early part of sequence.

Summary. We summarize our findings for LLMs beyond the three models discussed above. We also put other models into categories based on empirical observations.

- For feature dimensions, massive activations are consistently present in very few fixed dimensions.
- For sequence dimensions, we classify LLMs into three categories based on massive activations' locations:
  a) Starting token only.
     Models include LLaMA2-13B, MPT and GPT-2.
  b) Starting token and the first "strong" delimiter token (i.e., "`.`" or "`\n`")
     Models include LLaMA2-7B and LLaMA2-7B-Chat.
  c) Starting token, delimiter tokens (such as "`.`", "`\n`", "`'`" or "`,`"), and certain word tokens with weak semantics (such as "`and`", "`from`", "`of`" or "`2`" (Such numeric tokens exhibit massive activations only in certain contexts, e.g., dates and years. Refer to Figure 4.17 for an illustration on LLaMA2-70B.).
     Models include LLaMA2-70B, Mistral-7B, Mixtral-8x7B, Falcon-40B and Phi-2.

4.3.3 Difference from Outlier Features

With an understanding of the nature and locations of massive activations, we now discuss the differences between them and outlier features, a seemingly similar phenomenon in LLMs. [58] have identified the existence of outlier features characterized by large magnitudes within LLMs.

Conceptually, a massive activation is a scalar value, determined jointly by the sequence and feature dimensions; in contrast, *an outlier feature is a vector, corresponding to activations at all tokens*. Further, massive activations are present at extremely few tokens, while outlier features expect most activations in them to be large.

In practice, we find that massive activations do not overlap with outlier feature dimensions. We identify outlier features in LLaMA2-7B and 13B using the definition in [58]: a feature is deemed as an outlier feature if activation magnitudes exceed 6.0 at more than 25% of layers and 6% of tokens, on more than 90 out of 100 sequences. We discover 10 and 25 outlier features in these two models respectively. However, none of them correspond to the feature dimensions of massive activations.

4.4 The Role of Massive Activations in LLMs

While we have demonstrated the existence of massive activations and identified their locations, their functional role within LLMs is not yet clear. Are they important for internal computation? Or are they simply redundant activations with no effect? This section will delve deeper into LLMs to answer these questions. Different from the previous passive observations, we take a more proactive approach by inspecting how modifying massive activations affects the external behavior of LLMs.

We first measure the variances of massive activations across input sequences. Besides massive activations, we choose three other positions based on their average magnitudes, corresponding to the top 1%/10%, and the median within the hidden state. In Table 4.2, we show the mean and standard deviation of the activation values at these positions across 100 sequences, for LLaMA2-7B and 13B. We find that the variances of massive activations are considerably smaller relative to their mean values when compared to other activations.

We then modify the inference of LLMs by intervening massive activations at one layer—for a hidden state exhibiting massive activations, we manually set these activations to chosen fixed values. Then the altered hidden state is fed into the next layer, and the computation afterwards continues as normal. We modify massive activations in LLaMA2-7B and 13B. We evaluate the perplexity on WikiText, C4 and PG-19 and the mean zero-shot accuracy on BoolQ, PIQA, WinoGrande, Arc-Easy and Arc-Challenge. For each model, we perform the intervention once on the hidden state where massive activations first appear. This corresponds to layer 2 and layer 4 in LLaMA2-7B and 13B respectively.

Model	Top 1	Top 2	Top 1%	Top 10%	Median
LLaMA2-7B	2556.8 $\pm$ 141.0	-1507.0 $\pm$ 83.0	-0.14 $\pm$ 0.6	0.0 $\pm$ 0.5	0.2 $\pm$ 0.3
LLaMA2-13B	-1277.5 $\pm$ 14.6	-787.8 $\pm$ 8.0	0.9 $\pm$ 0.7	-0.3 $\pm$ 0.8	-0.3 $\pm$ 0.6

Table 4.2: The mean and variance of activation values at several positions, corresponding to the 2 largest, top 1% and 10%, and the median magnitudes within the hidden state. We find that the variation in massive activations is significantly lower in comparison to other activations.

Setting massive activations to zero. We evaluate the performance of LLMs without massive activations. We set their values to zero in the hidden state when they first appear, i.e., removing massive activations from intervened LLMs. The results (denoted by *Set to zero*) are shown in Table 4.3. Intriguingly, there is a significant degradation in model performance, e.g., exploding perplexity numbers. For comparative analysis, an equal number of activations—those with average magnitudes close to the median magnitude—are similarly set to zero. We find this leads to no performance drop. These results highlight the crucial role that massive activations play in the internal computation of LLMs.

Setting massive activations to mean values. We remove the small variances in the values of massive activations. Specifically, we adjust the values of massive activations to their empirical

	LLaMA2-7B				LLaMA2-13B			
Intervention	WikiText	C4	PG-19	Mean Zero-Shot	WikiText	C4	PG-19	Mean Zero-Shot
Original	5.47	7.85	8.57	68.95%	4.88	7.22	7.16	71.94%
Set to zero	`inf`	`inf`	`inf`	36.75%	5729	5526	4759	37.50%
Set to mean	5.47	7.86	8.59	68.94%	4.88	7.22	7.16	71.92%

Table 4.3: Intervention analysis of massive activations in LLaMA2-7B and 13B. We set massive activations to fixed values and evaluate the perplexity (↓) and zero-shot accuracy (%, ↑) of intervened models.

mean values. The means are computed on 100 sequences from RedPajama. The results of this intervention (denoted by *Set to mean*) are shown in Table 4.3. We find that there are negligible changes in perplexity and zero-shot accuracy. This shows that their values are constants and input agnostic, i.e., functioning similarly to bias terms.

To summarize our findings:

Massive activations act as fixed but important biases in LLMs.

Why these layers and tokens? The fact that these activations act as biases may explain why LLMs store them at certain layers and tokens:

- The tendency of these activations to appear at the starting token could be attributed to the fact that every autoregressive training instance contains an initial token. Since LLMs are based on next word prediction, the starting token is the only token used in all forward passes within a sequence.
- The existence of these activations in delimiter tokens might be due to the relatively low semantic value of these tokens, rendering them a low-cost option for storing such biases. Conversely, tokens with rich semantics would risk significant loss of input information, if they are repurposed to store biases.
- The fact that massive activations emerge only after a few initial layers may be because LLMs would require some initial layers to process the meaning of the tokens associated with massive activations. At these layers, their semantics may be transferred to other token positions via self-attention, and preserved moving forward.

4.5 Effects on Attention

In this section, we explore and study the internal mechanism of massive activations in LLMs, particularly in relation to self-attention.

4.5.1 Attention is Concentrated on Massive Activations

We observe a stark contrast in attention patterns when comparing layers before and after the appearance of massive activations in LLMs. Figure 4.5 shows the attention logits (before softmax),

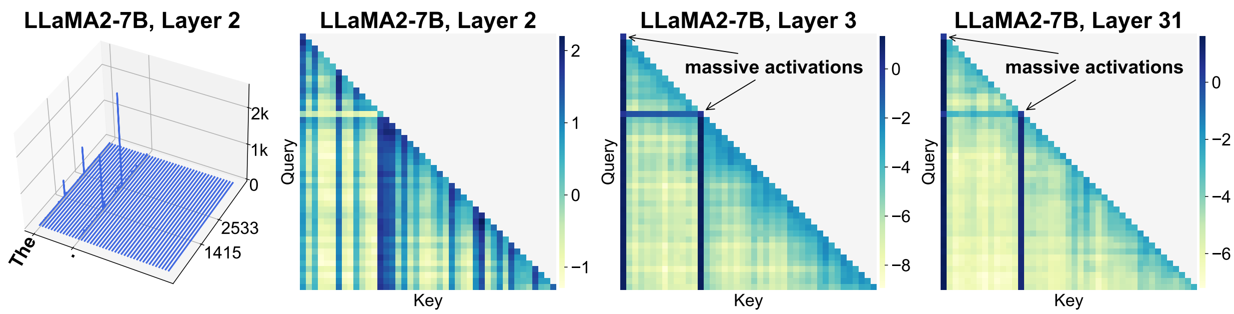


Figure 4.5: Attention patterns *before* and *after* massive activations appear in LLaMA2-7B. For each layer, we visualize average attention logits (unnormalized scores before softmax) over all heads, for an input sequence.

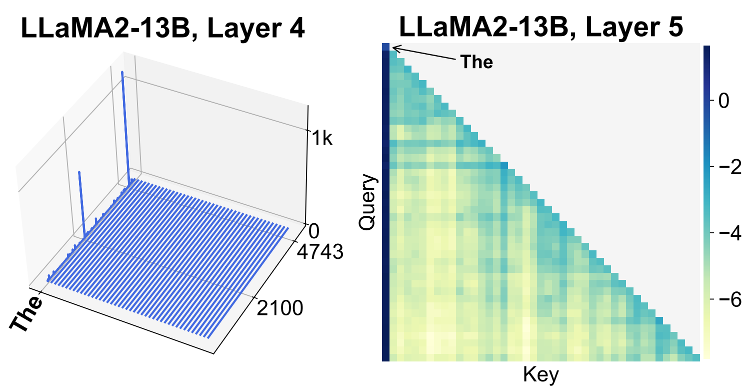


Figure 4.6: Attention patterns *after* massive activations emerge in LLaMA2-13B

averaged over all heads per layer in LLaMA2-7B. The input is a prompt from MMLU [88]: "*The following are multiple choice questions (with answers) about machine learning.*\n\n *...*". Recall that in LLaMA2-7B, massive activations first appear in the output of layer 2 (see Figure 4.4). We find that in layer 3 and deeper layers (e.g., layer 31), attention is mostly concentrated on the two tokens associated with massive activations. Our observations are also consistent across various LLMs. Figure 4.6 and Figure 4.7 demonstrate such attention concentration patterns in LLaMA2-13B and Phi-2, on the same input. See Chapter 4.8.5 for results on more LLMs.

We notice that there is a consistent pattern across models on the distribution of attention logit values. In Figure 4.5, Figure 4.6 and Figure 4.7, many attention logits tend to be negative following massive activations. They are mostly computed by the inner product between query and key states of tokens without massive activations. However, when the key states belong to tokens associated with massive activations, the resulting attention logits are slightly positive. Thus in the attention softmax (computed along each row), these special attention logits will attract most of the attention probability.

Recently, [190] showed that LLMs attend heavily to the starting token. Our findings on LLaMA2-13B in Figure 4.6 align with their results. Empirically, we find it is true for LLMs where massive activations are only found within the starting token. However, our results on LLaMA2-7B and Phi-2 indicate that LLMs also allocate substantial attention to other tokens and they are associated with massive activations. Furthermore, our results reveal a deeper cause for

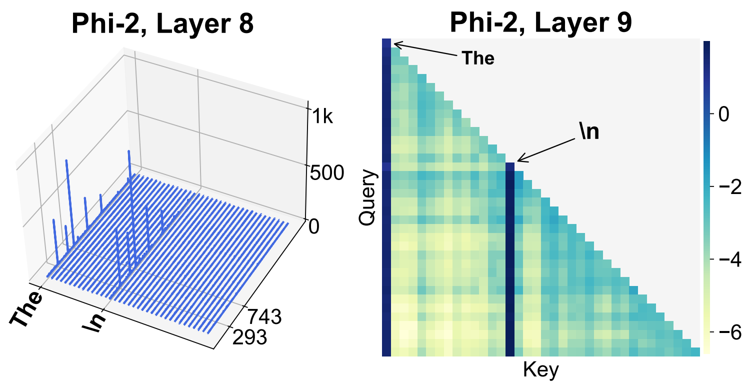


Figure 4.7: Attention patterns *after* massive activations emerge in Phi-2.

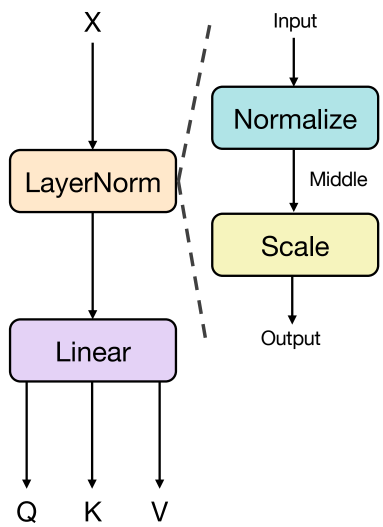


Figure 4.8: Attention LayerNorm and QKV linear projections.

the emergence of these attention concentration patterns.

4.5.2 Massive Activations Impose Implicit Attention Biases

In this part, we delve into the computation within the attention block and demonstrate that LLMs use massive activations to enforce an implicit bias term in self-attention.

Attention LayerNorm and QKV projections. We study the impact of massive activations on the query, key and value states (Q/K/V) in self-attention. In LLMs, at each layer, input features are processed by layer normalization[1] [31] and then transformed into query, key and value states via linear projections, as illustrated in Figure 4.8. This design choice is introduced in GPT-2 [2] and widely adopted in modern LLMs.

Figure 4.9 visualizes all hidden states computed in this schematic (LLaMA2-7B, layer 3). We find that at all stages, features of the two tokens associated with massive activations are drastically

[1] LLaMA2 uses a variant of layer normalization: RMSNorm [176].

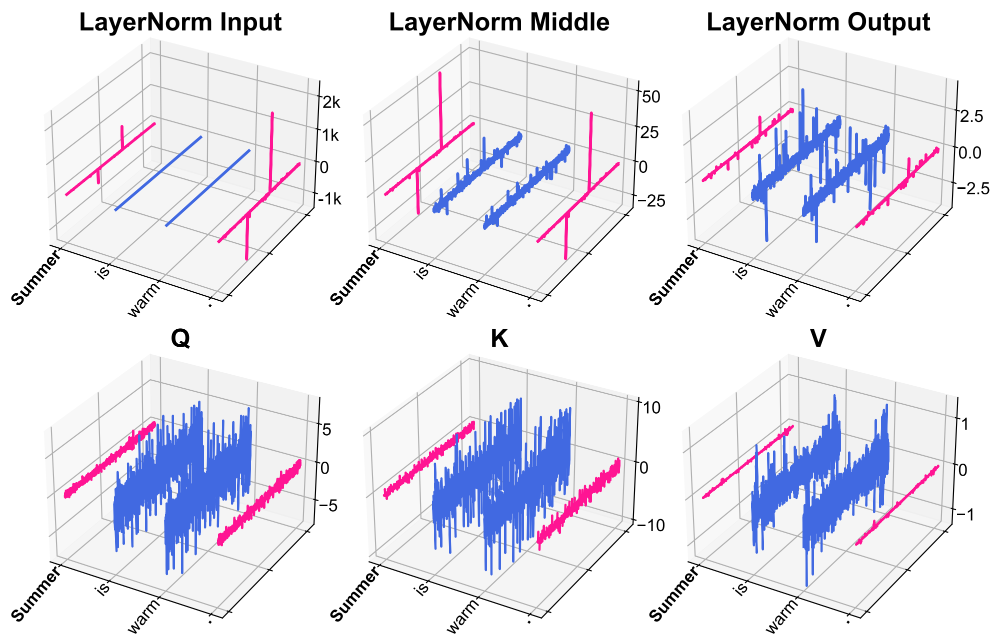


Figure 4.9: Layer 3, LLaMA2-7B. We highlight the embeddings of the two tokens where massive activations appear: the starting token and the period token.

different from other tokens. Specifically, after the first "normalize" step, the embeddings of these two tokens appear as a sparse vector with two distinct non-zero elements. Notably, the subsequent QKV states exhibit considerably smaller variations within each embedding. We hypothesize that the attention LayerNorm may play a pivotal role in this process (see Chapter 4.8.6 for further discussion).

Attention output decomposition. Given that attention is also concentrated on the tokens associated with massive activations (Section 4.5.1), we thus isolate these tokens and study their effects on the attention output (the layer of attention matrix multiplying value vectors). In

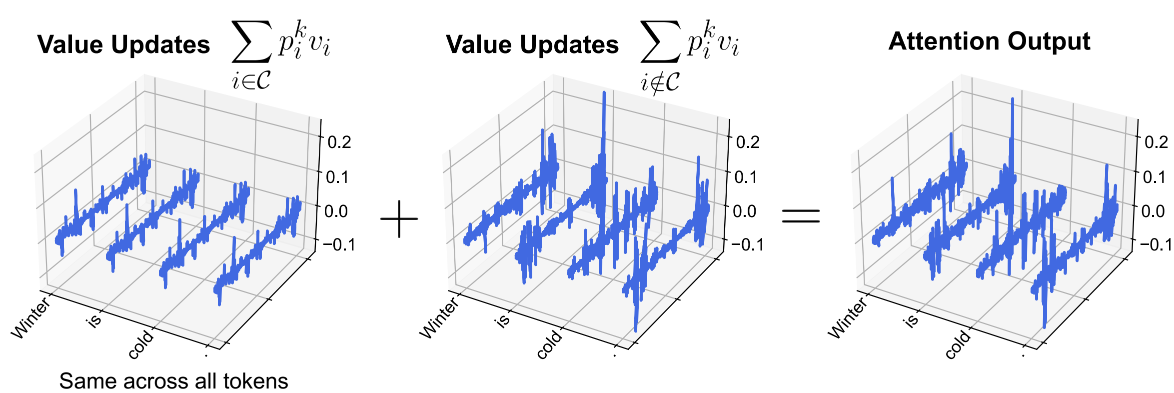


Figure 4.10: Value updates from tokens associated with massive activations are essentially the same.

Equation 4.2, we decompose the attention output at each token k into two parts: value updates from the tokens $\mathcal{C}$ where attention is concentrated; and value updates aggregated from other tokens.

$$\text{Attention}(Q, K, V)_k = \sum_{i \leq k} p_i^k v_i = \sum_{i \in \mathcal{C}} p_i^k v_i + \sum_{i \notin \mathcal{C}} p_i^k v_i \tag{4.2}$$

where p_i^k is the attention distribution of query token k to token i, and v_i is the value state of token i.

Figure 4.10 visualizes the decomposed value updates and the attention output in LLaMA2-7B, with the input prompt "*Summer is warm. Winter is cold.*". In this case, the set $\mathcal{C}$ consists of token `Summer` and the first period token. We can see that the value updates from $\mathcal{C}$ are nearly identical across tokens, i.e., they serve as additive bias terms, although not explicitly imposed. Furthermore, we note that this pattern of value update is strikingly similar across various inputs. We refer the reader to Chapter 4.8.7 for additional analysis. Overall, our results indicate that LLMs use massive activations to allocate substantial attention at certain tokens. These tokens are then utilized to form a constant bias term when computing the attention output.

4.5.3 Explicit Attention Biases Eliminate Massive Activations

Given the strong need of LLMs to learn implicit attention biases during pretraining, we thus experiment with directly augmenting self-attention with additional bias terms. Intriguingly, we find that models augmented with explicit attention biases do not exhibit massive activations.

Formulation. The idea is to model such attention biases explicitly, except not through repurposing existing tokens in the input sequence. Thus we introduce additional *learnable* parameters $\mathbf{k}', \mathbf{v}' \in \mathbb{R}^d$ for each head. Specifically, given input query, key and value matrices $Q, K, V \in \mathbb{R}^{T \times d}$, the augmented attention with explicit attention biases is computed as:

$$\text{Attention}(Q, K, V;\, \mathbf{k}', \mathbf{v}') = \text{softmax}\left(\frac{Q\left[K^T \;\; \mathbf{k}'\right]}{\sqrt{d}}\right)\begin{bmatrix} V \\ \mathbf{v}'^T \end{bmatrix} \tag{4.3}$$

where $\mathbf{k}'$ and $\mathbf{v}'$ are each concatenated with the key and value matrices K/V. The proposed attention can be used as a drop-in replacement of standard attention, without modifying other parts of Transformers, e.g., positional embeddings and MLP blocks.

Results. We train three GPT-2 models: the standard model, GPT-2 prepended with a sink token [190] and GPT-2 with explicit attention biases. See Chapter 4.8.8 for training setups. We find that the three models have the same performance at convergence but differ significantly in the status of massive activations, as demonstrated in Figure 4.11. Notably, in GPT-2 with explicit attention biases, massive activations disappear, as compared to the default GPT-2 and one with a sink token.

Figure 4.12 shows the three largest activation magnitudes at each layer. Notably, with explicit attention biases, top activation magnitudes in GPT-2 are increasing gradually as layers go deeper. These results indicate that explicit attention biases negate the necessity for LLMs to develop

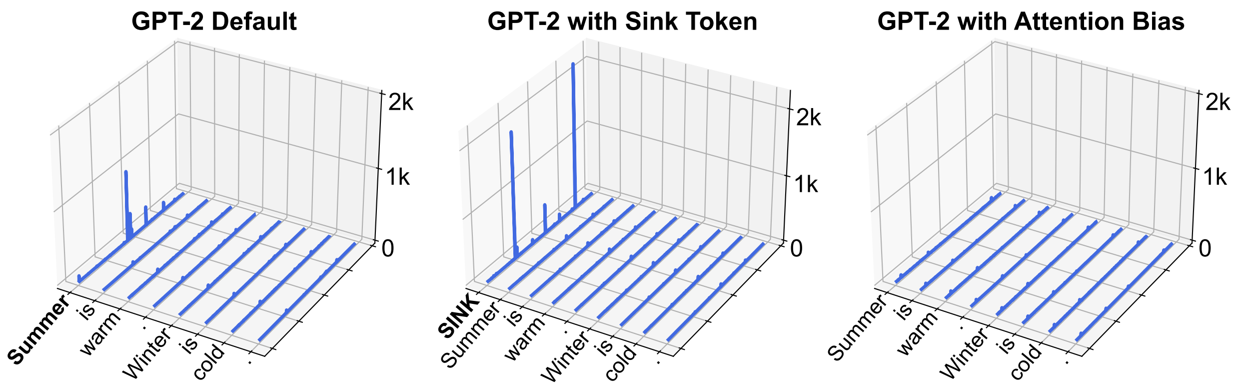


Figure 4.11: Massive activations disappear when training GPT-2 with explicit attention bias (Equation 4.3).

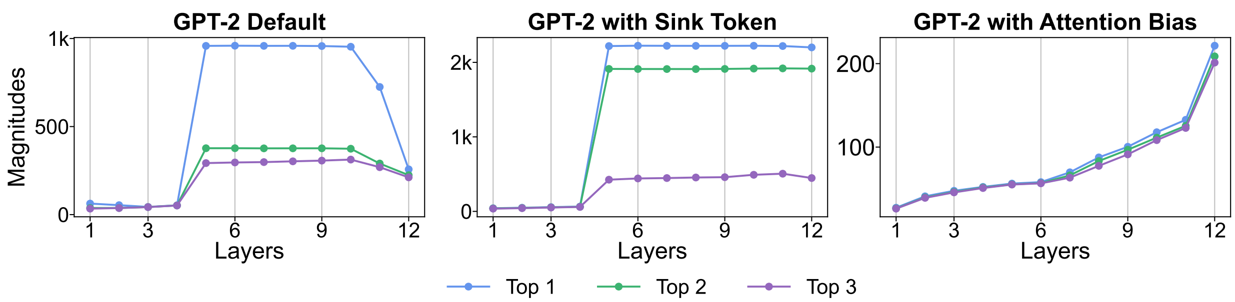


Figure 4.12: Three largest activation magnitudes in the output feature of each layer for three GPT-2 models.

massive activations during the pretraining phase. We leave it as future work to investigate other aspects of our alternative attention formulation, e.g. training stability [167].

To summarize our findings in this section:

Massive activations are connected to self-attention. LLMs use massive activations to concentrate substantial attention on very few tokens, injecting implicit bias terms in the attention computation. Further, massive activations can be eliminated by augmenting LLMs with explicit attention biases.

4.6 Massive Activations in Vision Transformers

In this section, we study if Vision Transformers (ViTs) [130] exhibit massive activations. We note that while ViTs and LLMs are both based on self-attention, ViTs employ global token mixing, which contrasts with the autoregressive nature of LLMs.

Massive activations in ViTs. We explore several model families based on ViTs: CLIP [197], MAE [193] and DINOv2 [192]. We examine the ViT-L models from these families. The activation magnitudes in the penultimate layer for an input image are illustrated in Figure 4.13. We find that massive activations exist in CLIP and DINOv2 ViT-L, where we highlight the corresponding

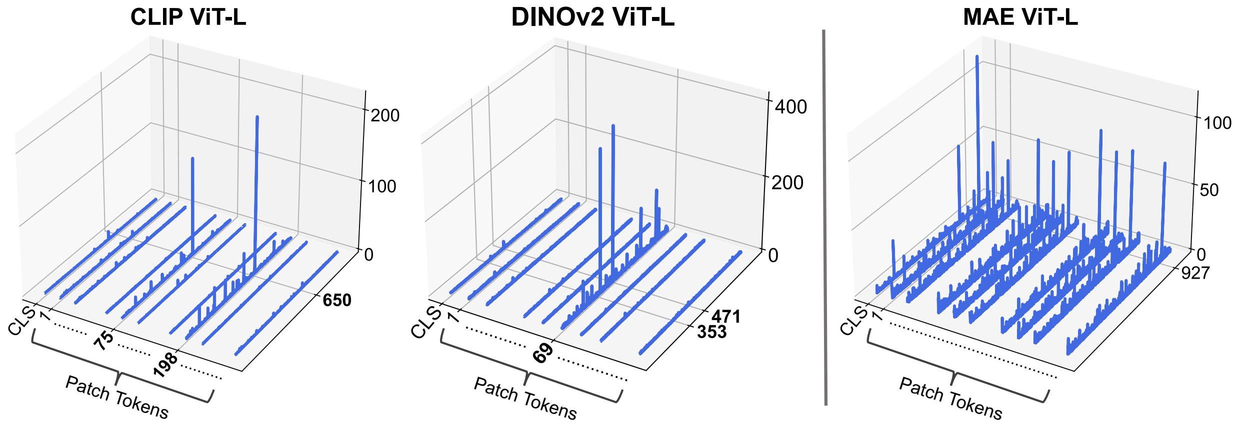


Figure 4.13: Massive activations are present in ViT-L from CLIP and DINOv2, but not MAE.

sequence dimensions. In these two models, there are extremely few activations (fewer than four) with significantly larger magnitudes than others. In addition, these activations are located in specific feature dimensions and appear in *random* patch tokens. However, we do not observe massive activations in MAE ViT-L. In this model, a feature dimension (927) exhibits uniformly large values across all tokens.

Massive activations are biases in ViTs. Figure 4.14 shows the three largest activation magnitudes and the median per layer in CLIP and DINOv2 ViT-L, averaged over 1k images. We find that massive activations are consistently present across images and their values remain largely the same around the mean values. It is worth noting that unlike LLMs, massive activations start to appear only in the later stages of ViTs.

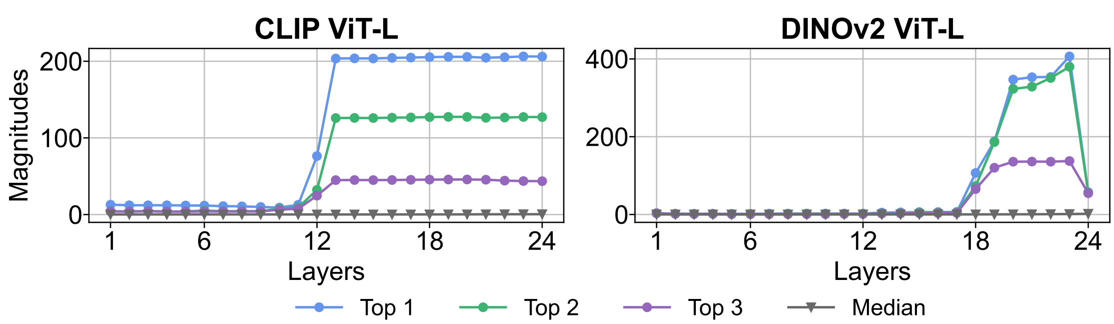


Figure 4.14: Three largest activation magnitudes and the median magnitude at each layer in CLIP and DINOv2 ViT-L.

Following our methodology in Section 4.4, we perform intervention analysis on CLIP ViT-L. We modify the two largest massive activations to zero and mean values respectively. The intervention is conducted on layer 13, where massive activations first appear within this model. Results are shown in Table 4.4, where we evaluate the zero-shot accuracy on ImageNet. We can see that setting massive activations to zero leads to significant drop in accuracy while setting to their means results in negligible accuracy drop. These results indicate that massive activations function as fixed but crucial biases in ViTs, aligned with our observations in Section 4.4.

CLIP ViT-L, layer 13	
Intervention	ImageNet acc (%)
Original	75.5
Set to zero	59.8
Set to mean	75.5

Table 4.4: Intervention analysis of massive activations in CLIP ViT-L.

Registers are biases in ViTs. Recently [191] propose to augment standard ViTs with additional learnable tokens, which they name as register tokens. They show that training ViTs with register tokens leads to smooth attention maps, and the resulting model family, namely DINOv2-reg, achieves superior downstream performance over DINOv2. Examining the largest ViT-G model in DINOv2-reg, we observe the existence of massive activations, as shown in Figure 4.15. However, different from standard ViTs, massive activations do not appear in patch tokens but exclusively within a fixed register token, i.e., register 3. This suggests that this model uses register 3 to store these activations. Figure 4.16 visualizes the attention distribution of the `[CLS]` token in the last layer. We find that most of the attention is allocated to register 3, echoing our previous findings in attention patterns (Section 4.5.1).

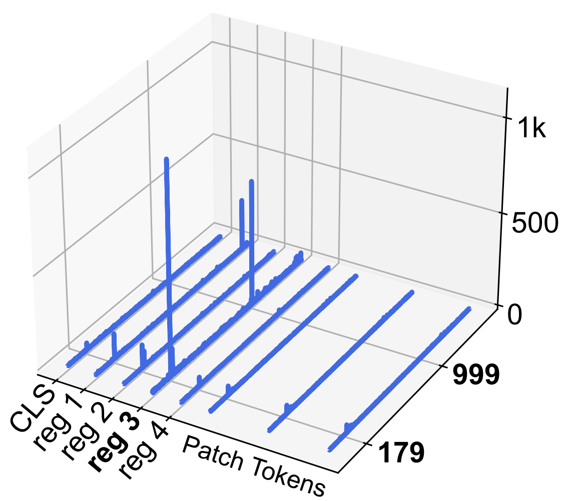


Figure 4.15: Massive activations in DINOv2-reg ViT-G.

	DINOv2-reg with 4 registers			
ImageNet acc (%)	ViT-S	ViT-B	ViT-L	ViT-G
Original	81.9	84.8	86.3	87.0
`Fix-Reg-Mean`	81.7	85.0	86.2	87.0

Table 4.5: We fix *all* register features at *every* layer to their means and evaluate the intervened ViTs.

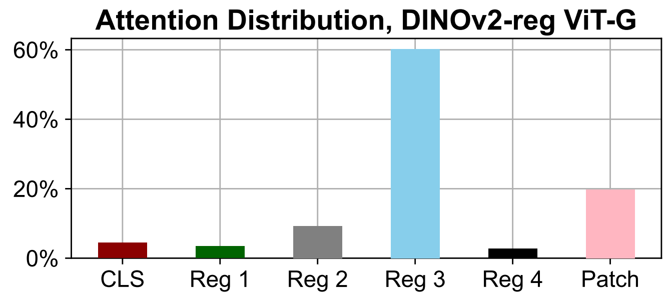


Figure 4.16: Average attention of the `[CLS]` token.

Further, we conduct intervention analysis to analyze the role of registers. We replace *all* register features at the output of *every* layer with their means, averaged over 10k ImageNet training images. This intervention removes the intended purpose of registers to aggregate global input information [191]. Table 4.5 shows the results. We find that ViTs with fixed register features achieve accuracy comparable to original models, suggesting that registers act as learned biases in ViTs. This leads to constant key and value states at register tokens, effectively introducing bias terms to self-attention (extra $\mathbf{k}'$ and $\mathbf{v}'$ in Equation 4.3). Thus a ViT with register tokens function equivalently to a standard ViT augmented with explicit attention biases.

To summarize our findings:

Massive activations exist in many but not all ViTs. Similar to those in LLMs, these activations act as constant biases. We also show the recently proposed register tokens have a similar function.

4.7 Related Work

Intriguing properties of autoregressive Transformers. [135] observed that in GPT-2's penultimate layer, there are feature dimensions containing activations with magnitudes up to 3,000. They found that these few dimensions dominate several standard measures for evaluating representation similarity. [163] found that the feature norm of the initial token in GPT-2 grows much faster than other tokens. [90] and [189] demonstrated the existence of outlier weights in the LayerNorm of GPT-2 and LLaMA2-13B and showed that setting them to zero leads to catastrophic drop in model performance. Notably, the feature dimension of this weight in LLaMA2-13B (i.e., 2100) corresponds to that of a massive activation (Figure 4.2).

Outlier features. Various existing works in quantization [57, 58, 150, 154, 184] have studied the existence of outlier features in LLMs. [58] showed that outlier features have large activation values in most of their sequence dimensions. While massive activations can be seemingly similar to outlier features, we discussed their fundamental differences in Section 4.3.3. More importantly, we show that massive activations can not be attributed to the existence of outlier features.

Attention concentration patterns. [195], [196] and [136] discovered that attention in BERT [18] tends to focus on the "separate" token `[SEP]`. [190] showed that LLMs assign most of the attention to the starting word token. [191] revealed the existence of attention artifacts in ViTs.

[183] found sparse activation patterns in ViTs that attract attention to certain tokens. Our work provides an in-depth analysis as to why these patterns emerge, specifically in relation to massive activations.

Biases in self-attention. There can be various notion of biases in the self-attention mechanism. First, simple additive bias terms can be used in linear layers for computing the query, key and value states [182]. Second, position biases can be inserted in self-attention to encode positional information of each token [180, 181]. There are also variants of biases with manually designed softmax operator [164, 187, 194]. Our work reveals that LLMs, even with standard self-attention formulation, would impose implicit bias components in the attention computation through massive activations.

4.8 Additional Results

4.8.1 Pretrained LLMs

In Section 4.3, we have demonstrated massive activations in LLaMA2-7B, LLaMA2-13B and Mixtral-8x7B. In this section, we evaluate more pretrained LLMs which cover a wide range of model families. We illustrate massive activations in LLaMA2-70B, LLaMA3 [25], Phi-2, Mistral-7B [173], MPT-7B [160] and Falcon-7B [159]. The results are presented in Figure 4.17, 4.18, 4.19, 4.20, 4.21, 4.22 and 4.23.

We make several observations. First, massive activations are consistently present in these models and they exhibit similar characteristics to those described in Section 4.3. Intriguingly, for LLaMA2-70B, we find that massive activations are found within tokens representing numerical values, e.g., token “0” and token “2”, as depicted in Figure 4.17. However, they do not appear in all numerical tokens (see the *rightmost* example in Figure 4.17). Another interesting finding is that the feature dimension of massive activations in both Mistral-7B (Figure 4.21) and Mixtral-8x7B (Figure 4.3) is identical (i.e., 2070), implying that the latter model may have been fine-tuned from the former.

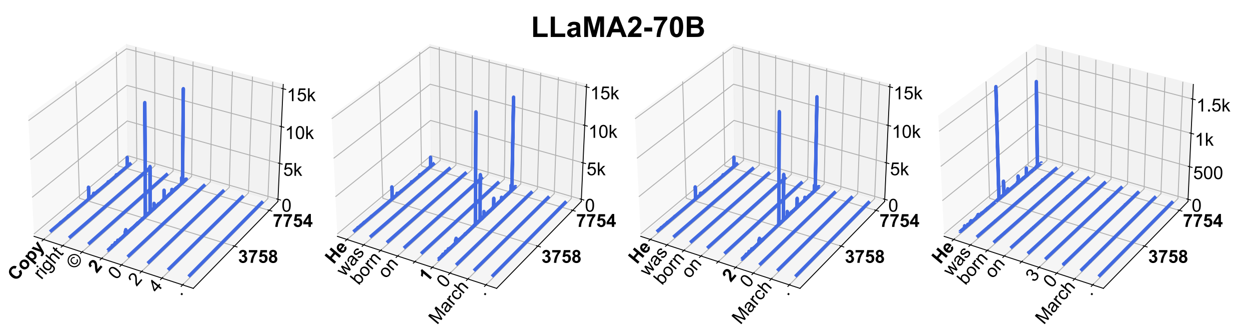


Figure 4.17: Massive activations in LLaMA2-70B.

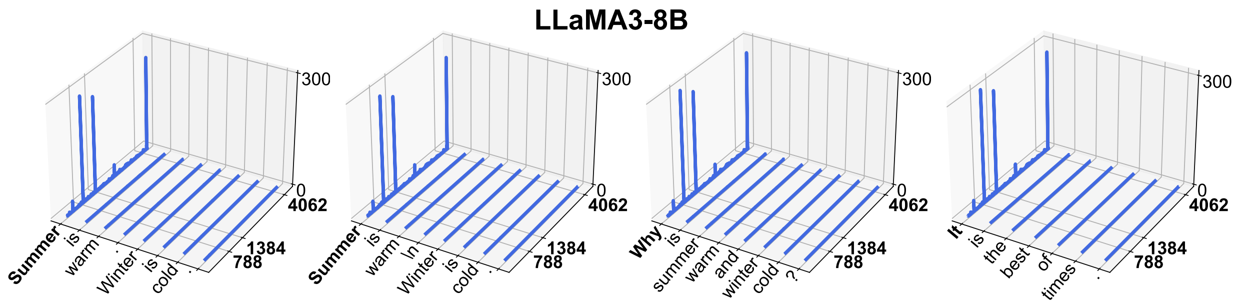


Figure 4.18: Massive activations in LLaMA3-8B.

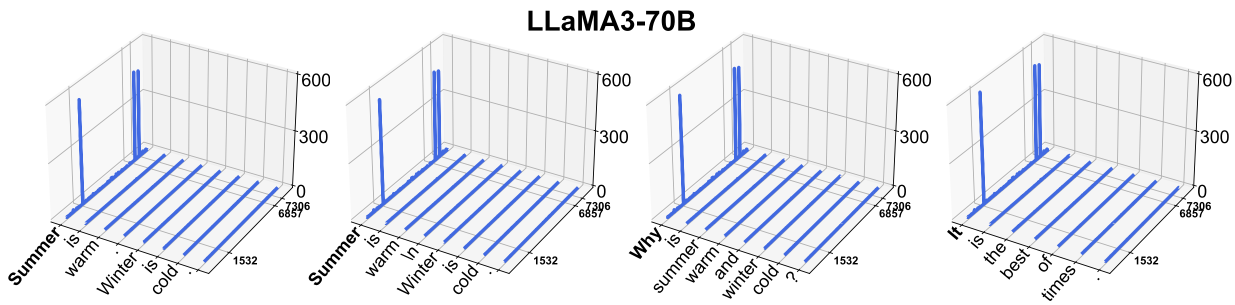


Figure 4.19: Massive activations in LLaMA3-70B.

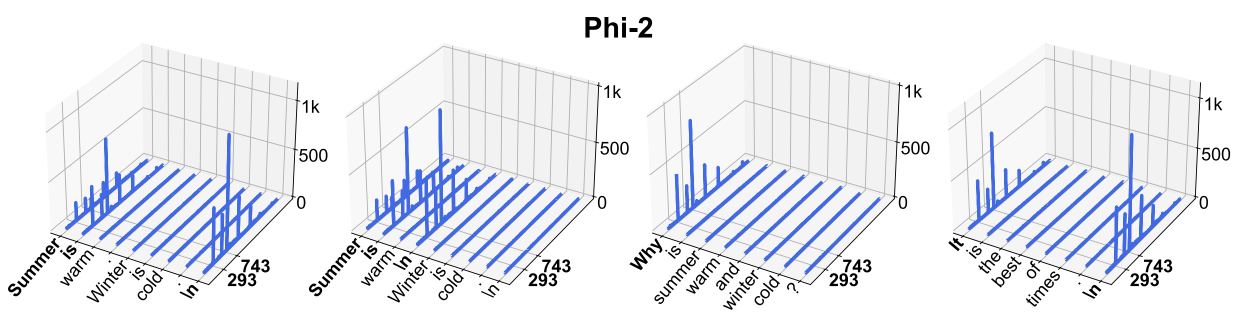


Figure 4.20: Massive activations in Phi-2.

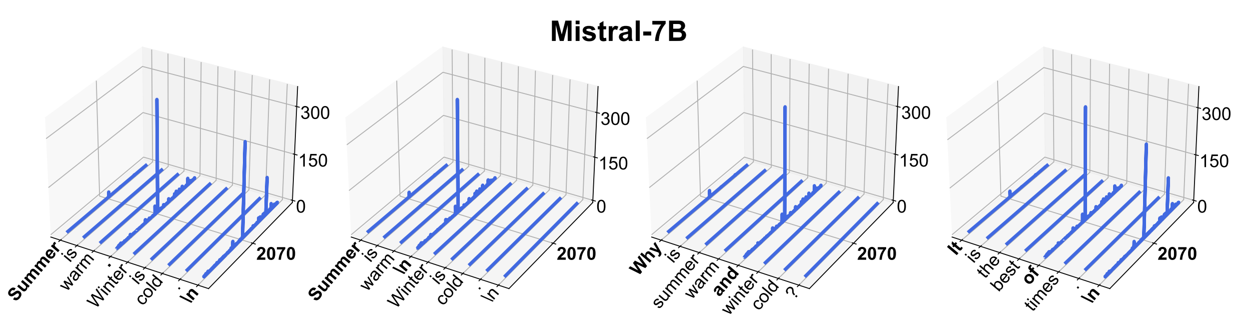


Figure 4.21: Massive activations in Mistral-7B.

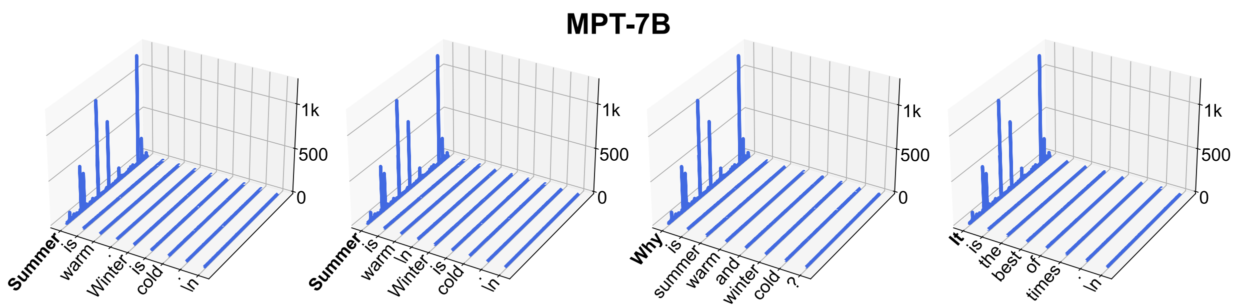


Figure 4.22: Massive activations in MPT-7B.

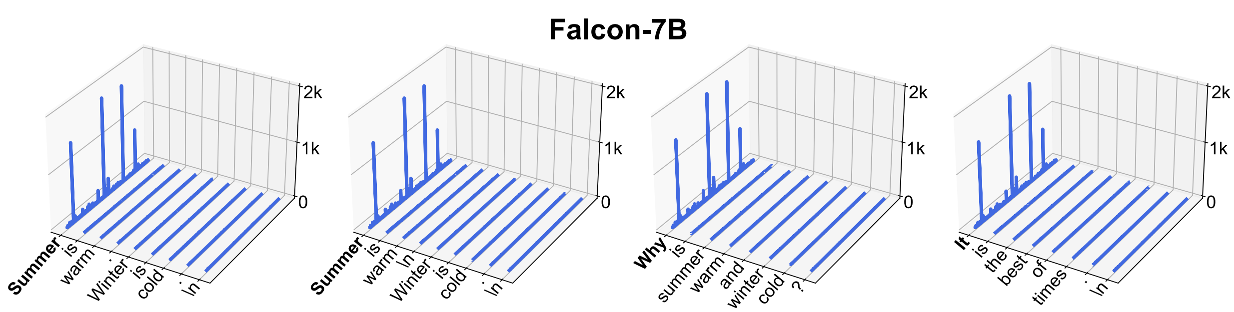


Figure 4.23: Massive activations in Falcon-7B.

4.8.2 Fine-tuned LLMs

Our results so far are focused on pretrained LLMs. However, a significant application of LLMs lies in their use for chat purposes. Instruction fine-tuning [26] is essential for developing models capable of generating coherent responses to questions. In this part, we demonstrate massive activations in these fine-tuned models. We evaluate fine-tuned models from models in LLaMA2 and Mistral. The results are shown in Figure 4.24, 4.25, 4.26 and 4.27.

We can see that massive activations persist after instruction fine-tuning. Moreover, the values and positions of massive activations remain largely the same as the original pretrained LLMs. For LLaMA2-7B, this can be seen by comparing Figure 4.24 and Figure 4.1. However, one exception is Mixtral-8x7B. We find that massive activations disappear from the newline token “\n” after fine-tuning, as shown by comparing Figure 4.27 and Figure 4.3. We leave the study on how instruction fine-tuning affects massive activations for future work.

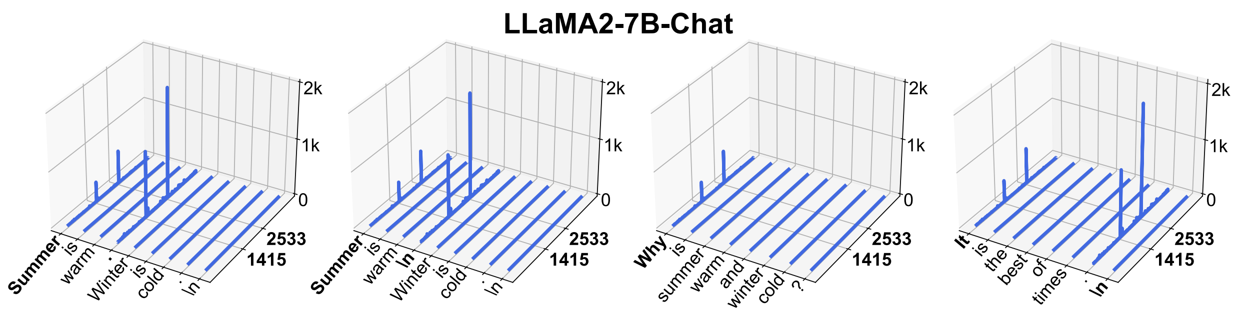


Figure 4.24: Massive activations in LLaMA2-7B-Chat.

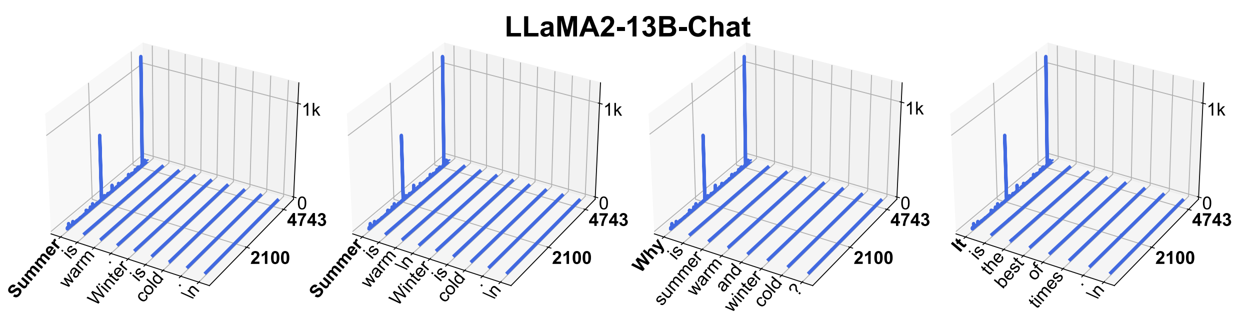


Figure 4.25: Massive activations in LLaMA2-13B-Chat.

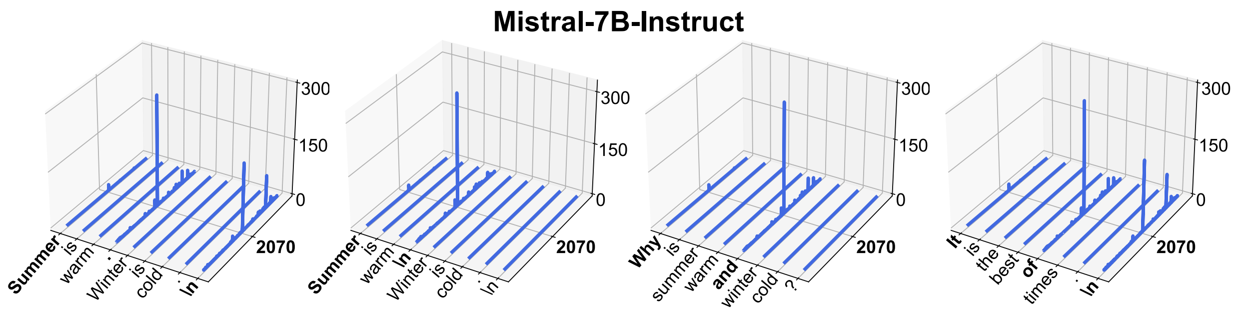


Figure 4.26: Massive activations in Mistral-7B-Instruct.

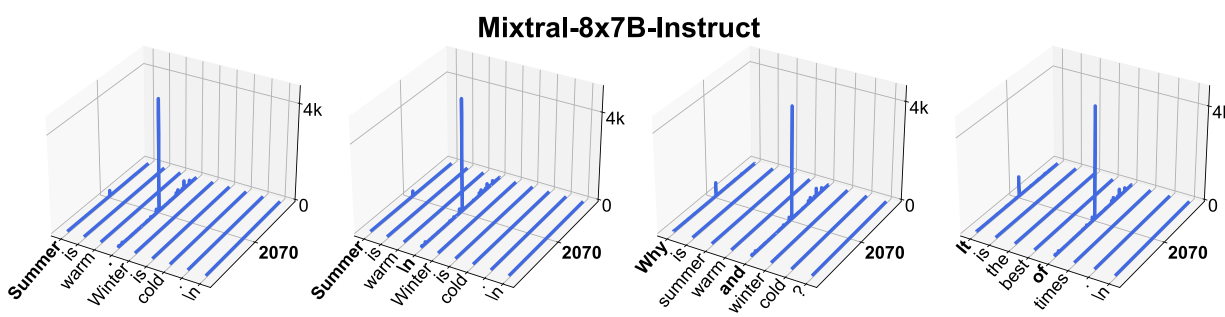


Figure 4.27: Massive activations in Mixtral-8x7B-Instruct.

4.8.3 BOS Token <s>

In some tokenizers, e.g., LLaMA2, the BOS token <s>, also known as the beginning of sequence token, can be prepended to the input sequence. For the experiments presented in other parts of the paper, we turn off this option, where all sequences do not start with the BOS token.

In Figure 4.28, 4.29 and 4.30, we show massive activations in LLaMA2-7B, LLaMA2-13B and Mixtral-8x7B, with the same input sequences as in Section 4.3. We find that massive activations persist with a prepended BOS token. In LLaMA2-7B and LLaMA2-13B, the locations of massive activations, i.e., sequence and feature dimensions, are not altered. However, for Mixtral-8x7B, some massive activations shift to the BOS token <s>. We leave the study on how the BOS token <s> affects the positions of massive activations for future work.

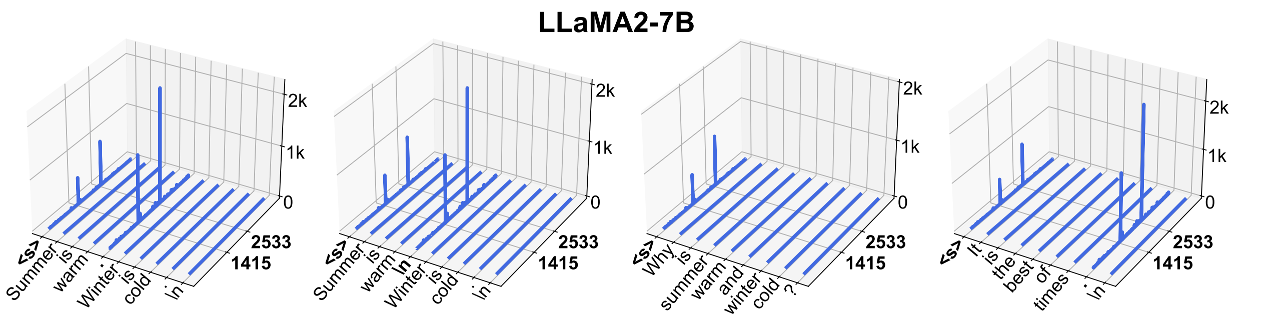


Figure 4.28: Massive activations in LLaMA2-7B when the input is prepended with a BOS token <s>.

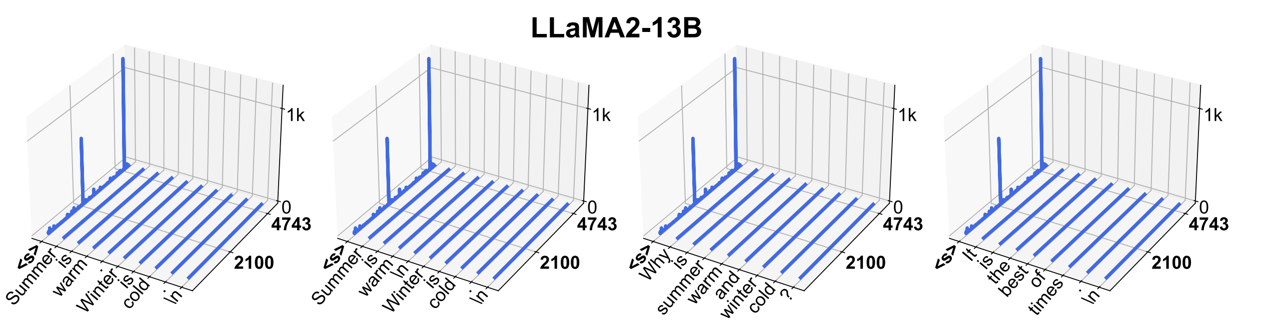


Figure 4.29: Massive activations in LLaMA2-13B when the input sequence is prepended with a BOS token <s>.

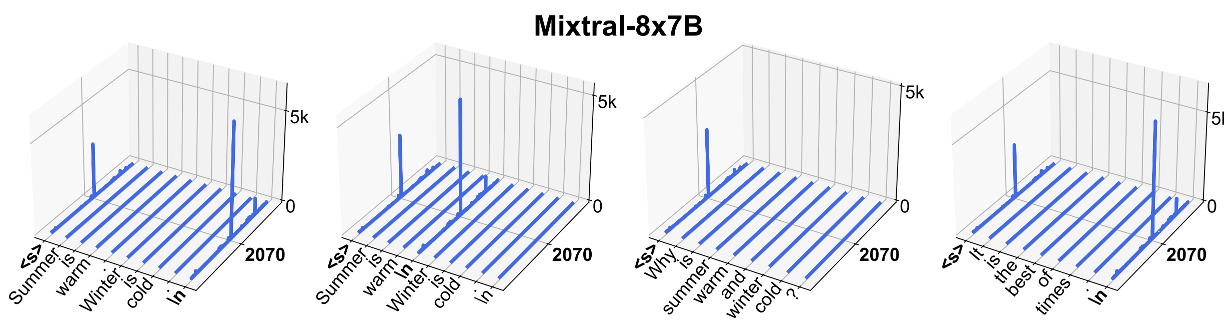


Figure 4.30: Massive activations in Mixtral-8x7B when the input sequence is prepended with a BOS token `<s>`.

4.8.4 Layer-Level Analysis

In Section 4.3.1, we have presented the layer-level analysis results for LLaMA2-7B, LLaMA2-13B and Phi-2. In Figure 4.31, we provide the comprehensive results for all LLMs examined in this paper (listed in Table 4.7). This includes LLMs from LLaMA2, Mistral, MPT, Falcon, OPT and GPT-2 model families. For each model, we show the three largest activation magnitudes as well as the median at each layer.

We can see that the trend of massive activations we observe in Section 4.3.1 holds true for LLMs in general. Massive activations tend to remain constant in most of the intermediate layers. They emerge in the early layers and disappear in the last layer.

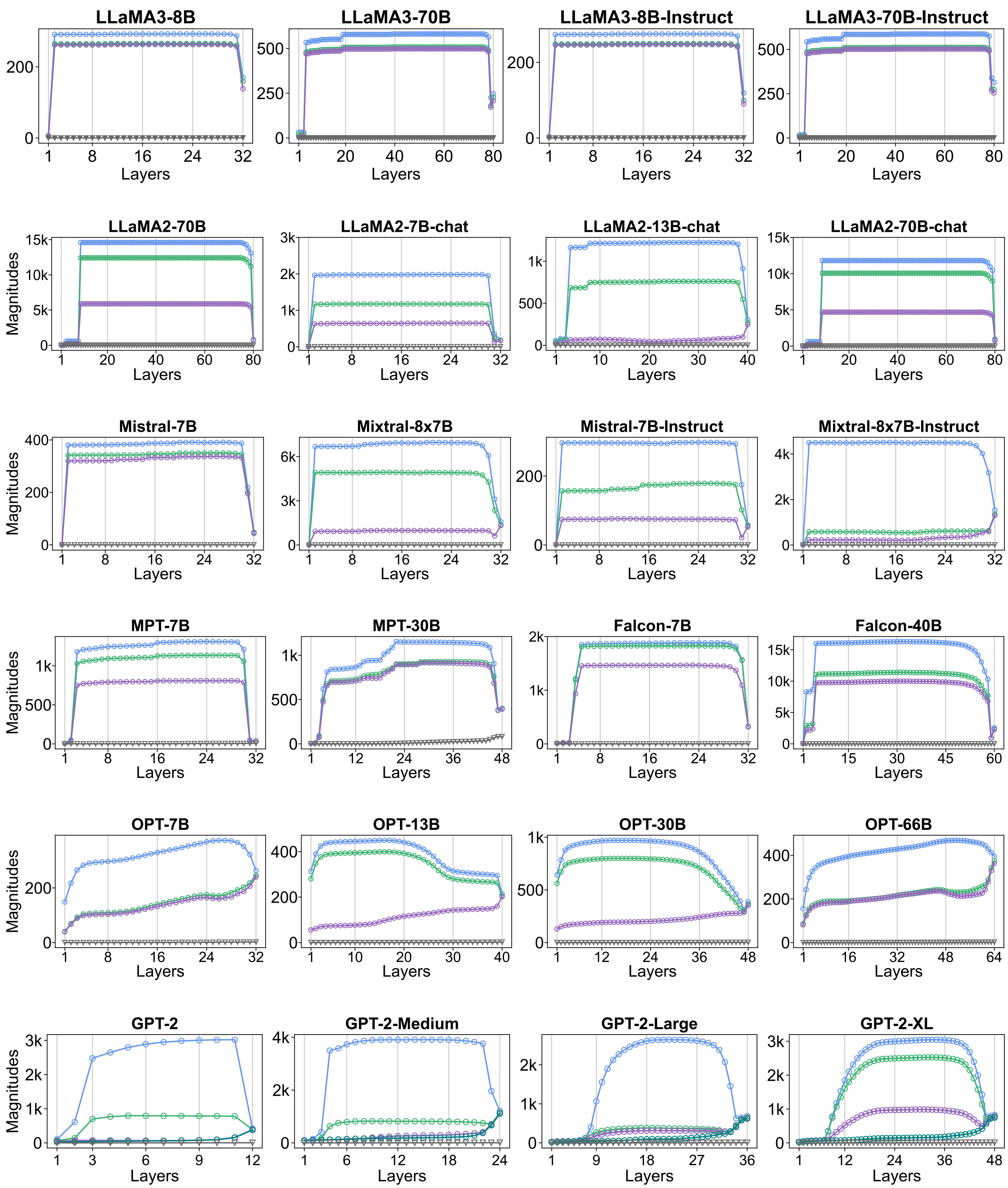


Figure 4.31: Layer-level analysis of LLMs. For each model, we show the three largest activation magnitudes as well as the median per layer.

4.8.5 Attention Concentration on Massive Activations

In Section 4.5, we have demonstrated the attention concentration pattern in LLaMA2-7B, LLaMA2-13B and Phi-2. We now illustrate this phenomenon for more LLMs. Figure 4.32 and Figure 4.33 show the results for LLaMA2-70B and Mistral-7B. For these two models, massive activations are formed in the output feature of layer 9 and layer 2 respectively.

We can see that attention is predominantly focused on the sequence dimensions of massive activations. In the case of LLaMA2-70B, as depicted in Figure 4.32, massive activations are found in the starting word token and also token `2`. These two tokens receive substantial attention logits. Additionally, we visualize the attention probability in Figure 4.34, 4.35 and 4.36. The attention softmax is computed along each row, thus resulting in these special tokens being allocated a much higher attention probability.

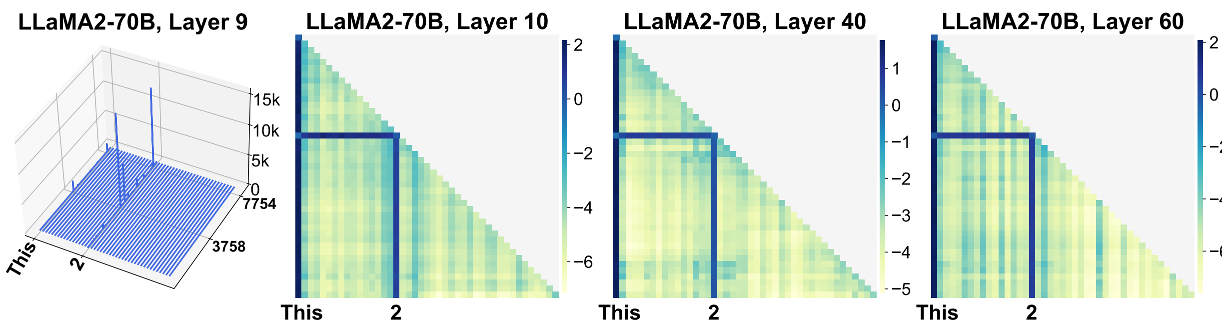


Figure 4.32: Average attention logits over all heads in layers 10, 40 and 60 of LLaMA2-70B. The input sequence is "*This book, including all illustrations and text, is protected under Copyright©2024 and may not be reproduced or transmitted in any form without the prior written permission of the copyright owner.*".

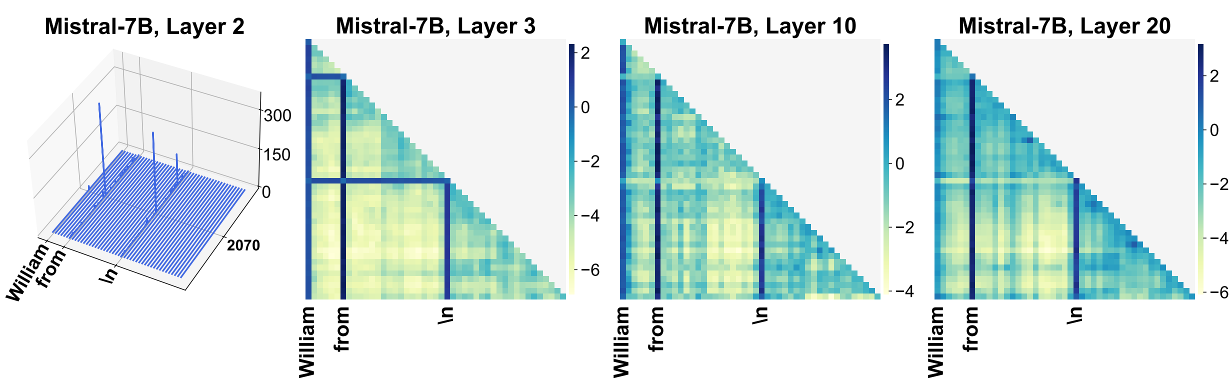


Figure 4.33: Average attention logits over all heads in layers 3, 10 and 20 of Mistral-7B. The input sequence is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.*$\backslash n$ *His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*".

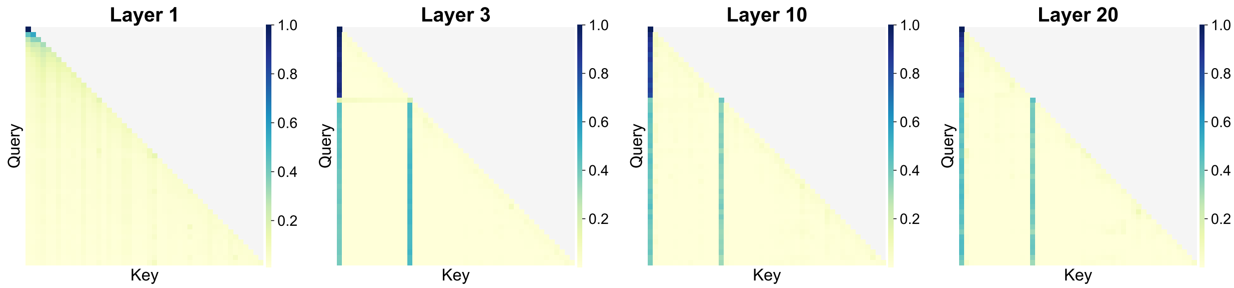


Figure 4.34: Average attention probability over all heads in intermediate layers of Llama2-7B. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.*$\backslash n$ *His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*".

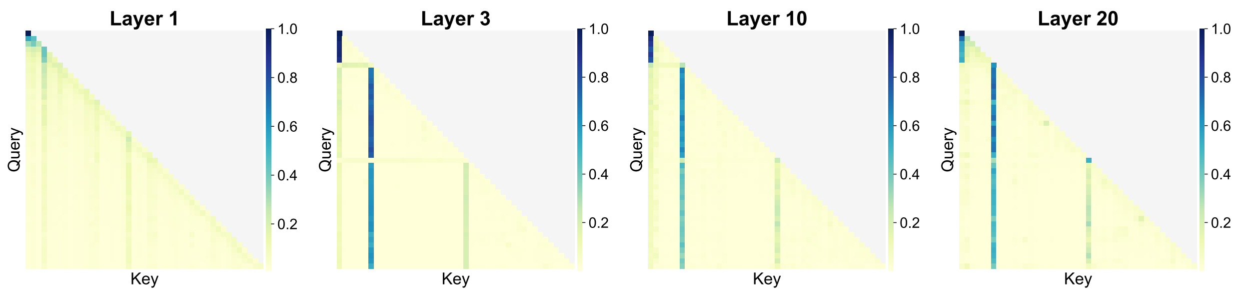


Figure 4.35: Average attention probability over all heads in intermediate layers of Mistral-7B. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.*$\backslash n$ *His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*".

4.8.6 Attention LayerNorm

Our analysis in Section 4.5.2 indicates that tokens associated with massive activations have drastically different key and value states. In this part, we investigate how attention layernorm plays a crucial role in this process.

Preliminaries. There are two specific types of layer normalization commonly used in LLMs. One is the standard layer normalization [31]. Suppose we have a feature vector $x \in \mathbb{R}^d$, LayerNorm will normalize this feature to fix the mean and variance and then re-scale with element-wise affine transformation:

$$\bar{x}_i = \frac{x_i - \mu}{\sigma} * g_i + b_i, \qquad \text{where } \mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}. \tag{4.4}$$

where $g, b \in \mathbb{R}^d$ are parameters of the affine transform, also called the gain and bias.

In addition to the original LayerNorm, a variant of layer normalization has also been used in LLaMA2 and Mistral models. Specifically, Root Mean Square Normalization (RMSNorm) [176]

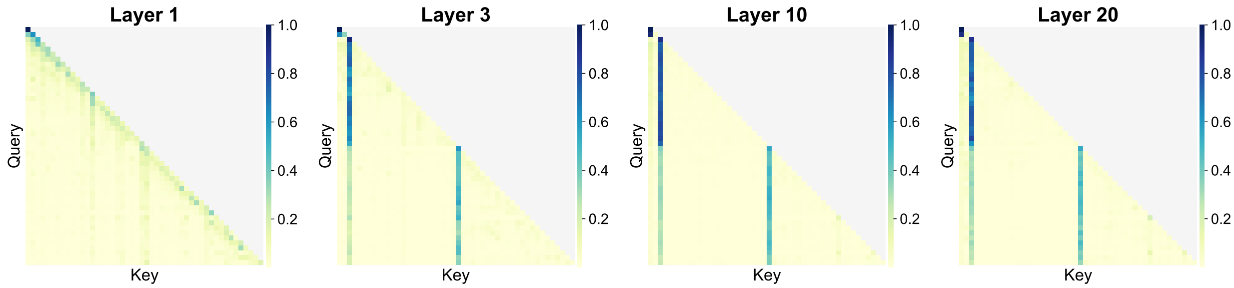


Figure 4.36: Average attention probability over all heads in intermediate layers of Phi-2. The input prompt is "*William Shakespeare was a famous writer from England who wrote plays and poems. He is considered one of the best writers ever.$\backslash n$ His works include famous plays like 'Romeo and Juliet' and 'Hamlet'.*".

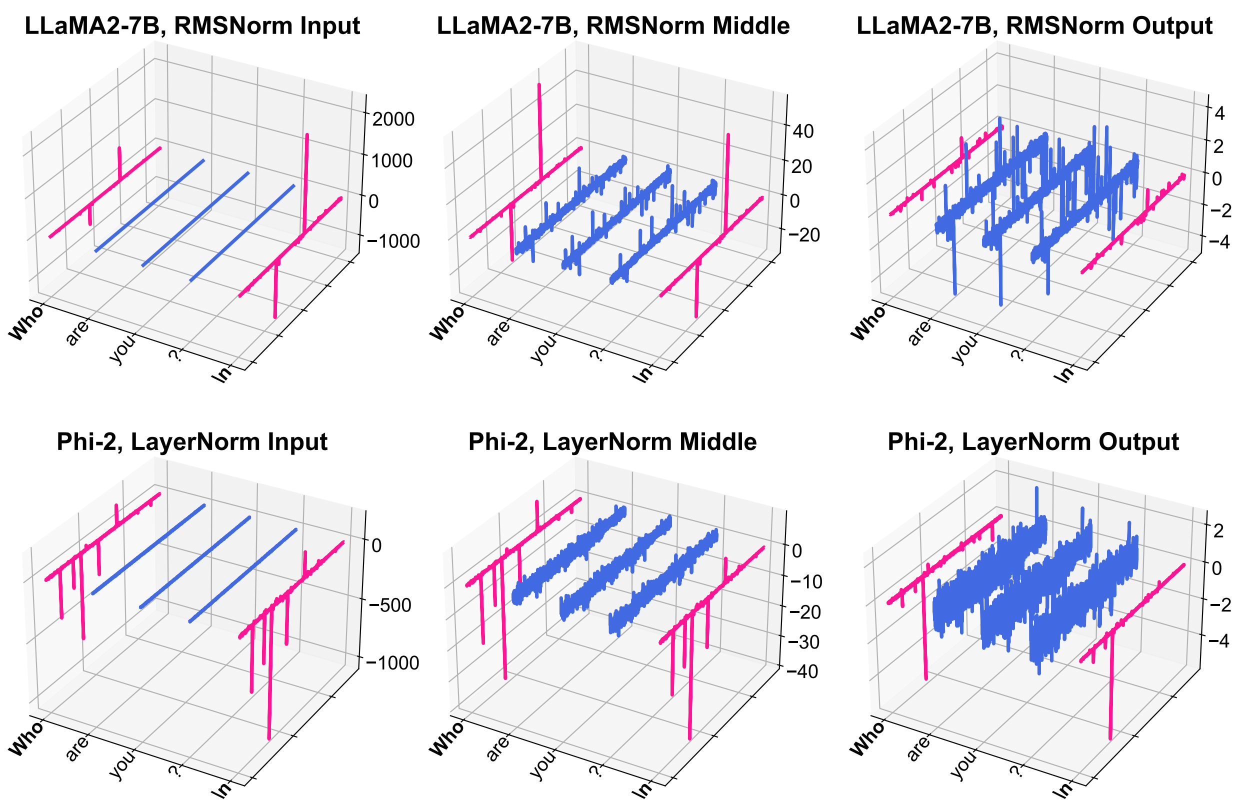


Figure 4.37: Activation trajectory in the attention LayerNorm of LLaMA2-7B and Phi-2, where the LayerNorm input contains massive activations. Note that LLaMA2-7B uses a variant of layer normalization: RMSNorm [176] and Phi-2 uses the default LayerNorm [31].

normalizes the feature $x \in \mathbb{R}^d$ with the root mean square (RMS) statistic:.

$$\bar{x}_i = \frac{x_i}{\text{RMS}(\mathbf{a})} * g_i, \quad \text{where } \text{RMS}(\mathbf{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}. \tag{4.5}$$

where $g \in \mathbb{R}^d$ is the gain parameter.

For both LayerNorm and RMSNorm, when there are a few activations in $x \in \mathbb{R}^d$ that have significantly large magnitudes, the denominator in the normalization step, i.e., σ in Equation 4.4 and $\mathrm{RMS}(\mathbf{x})$ in Equation 4.5, becomes large as a result. In fact, the denominator is almost determined by these few massive activations. The large denominator will push all normal values to zero while preserving the outlier nature of massive activations. This will effectively create a drastically different normalized feature, determined by the few massive activations. Figure 4.37 shows two activation trajectory in both RMSNorm and LayerNorm. We can see that how the normalization step (*middle*) preserves the outlier activations in tokens `Who` and `\n` and the normalized features at these two tokens become extremely similar.

4.8.7 Implicit Attention Biases

In Section 4.5.2, we have shown how the value updates from the tokens associated with massive activations tend to be largely identical. Here we extend those findings by examining additional input prompts and layers within the LLaMA2-7B model. We use four input prompts: "*Are you cold?\n Grab a jacket.*", "*Will it snow?\n Check the forecast.*", "*Did she call?\n I missed it.*" and ""*I am doing well. Thank you for asking.*"". We visualize the value updates in layer 3, layer 15 and layer 30 in Figure 4.38, Figure 4.39 and Figure 4.40 respectively. We focus on the latter half of the input sequence, following the two tokens associated with massive activations. We can see that in the same layer, the value updates $\sum_{i \in C} p_i^k v_i$ display remarkable similarity across the different input sequences.

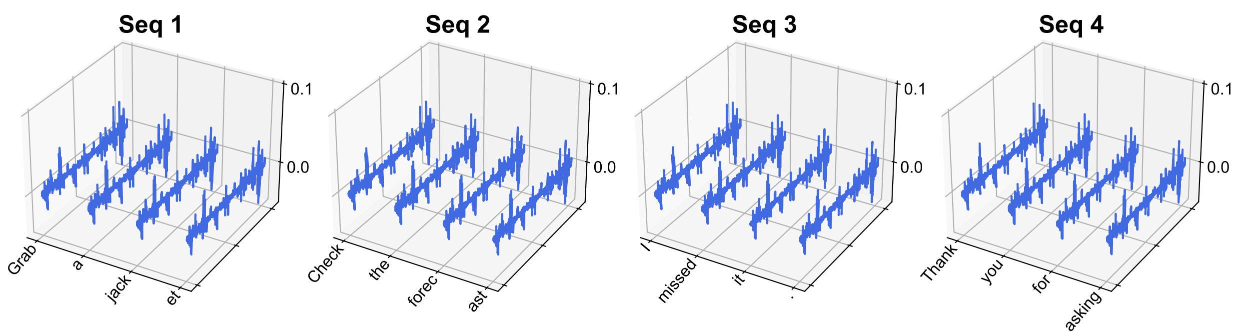


Figure 4.38: Value updates $\sum_{i \in C} p_i^k v_i$ at layer 3 of LLaMA2-7B, with four input sequences.

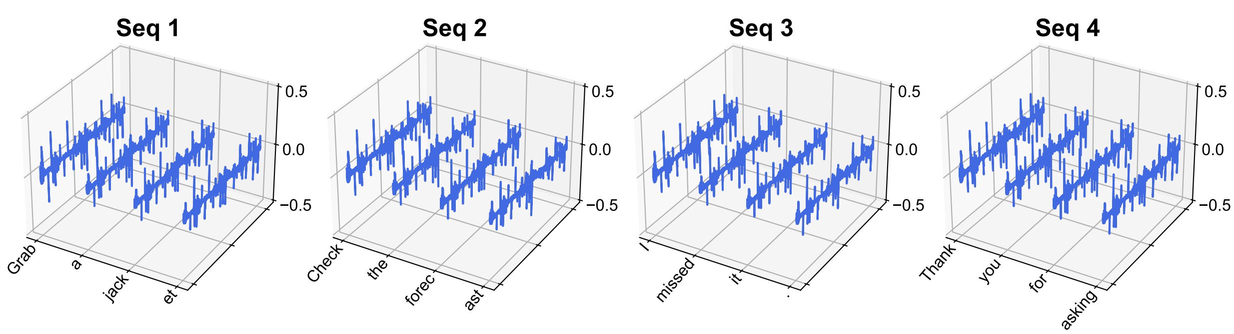


Figure 4.39: Value updates $\sum_{i \in C} p_i^k v_i$ at layer 15 of LLaMA2-7B, with four input sequences.

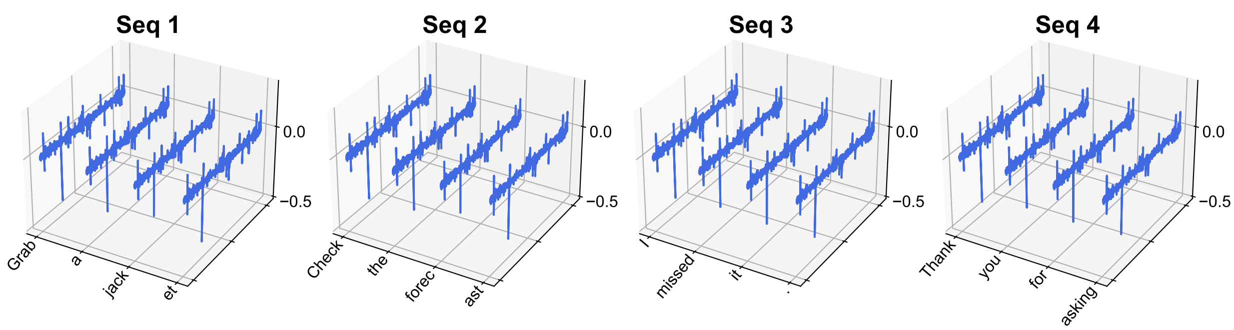


Figure 4.40: Value updates $\sum_{i \in \mathcal{C}} p_i^k v_i$ at layer 30 of LLaMA2-7B, with four input sequences.

4.8.8 Explicit Attention Biases

Experimental setup. We use the open-source reproduction of GPT-2 from the NanoGPT repository [165]. We use the default recommended training setup and optimizer setting. For each of the three GPT-2 models, we train for 50,000 iterations, with a total of approximately 2B tokens. For the GPT-2 with a sink token, we follow [190], where we prepend each training sequence with a learnable sink token `[SINK]`. When computing the training loss, we do not include the cross-entropy loss computed on the prepended sink token. For GPT-2 with explicit attention biases, we initialize each $\mathbf{k}'$ and $\mathbf{v}'$ with $\mathcal{N}(\mathbf{0}, 0.02\mathbf{I})$.

Results. Regarding the performance of the three GPT-2 models we evaluate in Section 4.5.3, we find that after 50,000 training iterations, they have the same perplexity on the validation split constructed from OpenWebText2 [44]: 3.04.

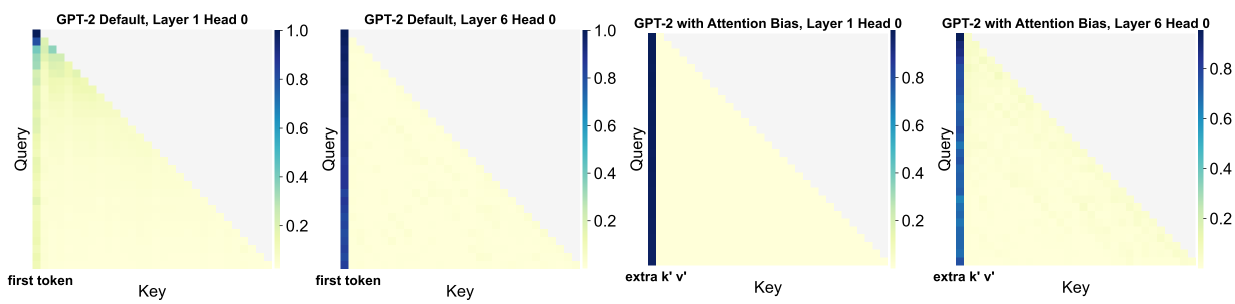


Figure 4.41: Attention distribution in default GPT-2 and GPT-2 with explicit attention bias.

In Figure 4.41, we visualize the attention distribution in both default GPT-2 and GPT-2 with explicit attention biases, where we plot the average attention probability over 50 sentences each with 30 tokens. First, we find that our observations on the relationship between massive activations and attention concentration hold for the default GPT-2 model. Second, for the GPT-2 model with explicit attention bias, most of the attention probability is assigned to the extra $\mathbf{k}'$ and $\mathbf{v}'$ vectors we inserted. Intriguingly, this also holds for initial layers as well (e.g., layer 1), suggesting the strong need for LLMs to form this attention concentration pattern during pretraining.

We also experiment with other ways of injecting biases in the self-attention computation:

1. The first one is a special case of our proposed formulation in Equation 4.3, where both $\mathbf{k}'$ and $\mathbf{v}'$ are zero vectors. Equation 4.6 shows the computation of this variant of self-attention. This is also equivalent to the previous proposed Softmax-off-by-one [164].

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q\begin{bmatrix}K^T & \mathbf{0}\end{bmatrix}}{\sqrt{d_k}}\right)\begin{bmatrix}V \\ \mathbf{0}^T\end{bmatrix} \tag{4.6}$$

2. Since Equation 4.3 can be viewed as inserting a sequence dimension, we also experiment with inserting one extra feature dimension. Specifically, we add learnable parameters $\mathbf{q}', \mathbf{k}' \in \mathbb{R}^T$ and concatenate them with the query and key states respectively. This variant of self-attention is as follows:

$$\text{Attention}(Q, K, V; \mathbf{q}', \mathbf{k}') = \text{softmax}\left(\frac{\begin{bmatrix}Q & \mathbf{q}'\end{bmatrix}\begin{bmatrix}K & \mathbf{k}'\end{bmatrix}^T}{\sqrt{d_k}}\right)V \tag{4.7}$$

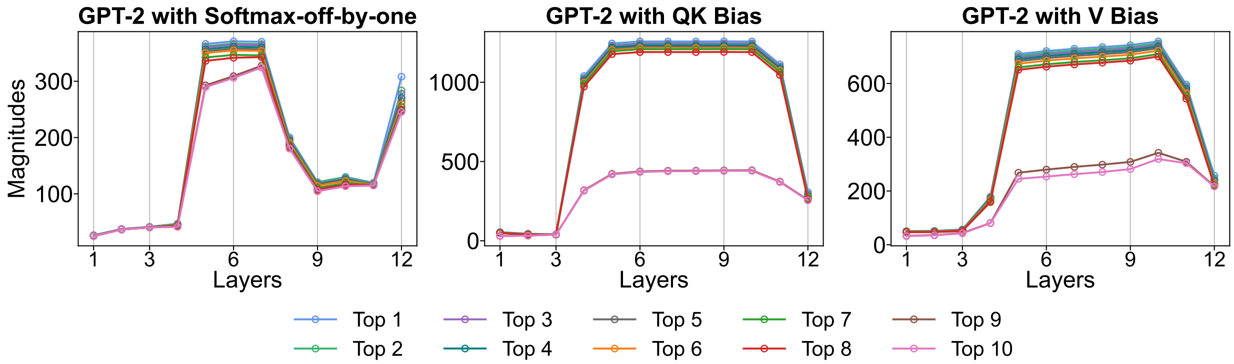


Figure 4.42: Ten largest activation magnitudes at each layer in three GPT-2 models.

3. We also experiment with a simple way to enforce constant value updates by injecting an extra value parameter $\mathbf{v}' \in \mathbb{R}^{d_k}$. This variant of self-attention is as follows:

$$\text{Attention}(Q, K, V; \mathbf{v}') = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V + \mathbf{v}' \tag{4.8}$$

Figure 4.42 visualizes the ten largest activation magnitudes in three GPT-2 models, corresponding to the three formulations of biases in Equation 4.6, 4.7 and 4.8. We find that these alternatives are not able to eliminate massive activations during pretraining.

4.8.9 Massive Activations in ViTs

We present results of massive activations in ViTs on 4 images from Figure 4.43. Results of CLIP ViT-L, DINOv2 ViT-L and DINOv2-reg ViT-G are shown in Figure 4.44, Figure 4.45 and Figure 4.46. We highlight the patch tokens exhibiting massive activations. For standard ViTs like CLIP ViT-L and DINOv2 ViT-L, massive activations appear in random patch tokens, i.e. the sequence dimensions of massive activations vary across input images. For DINOv2-reg ViT-G, they exist in a fixed register token, i.e., register 3.

Figure 4.43: Example images.

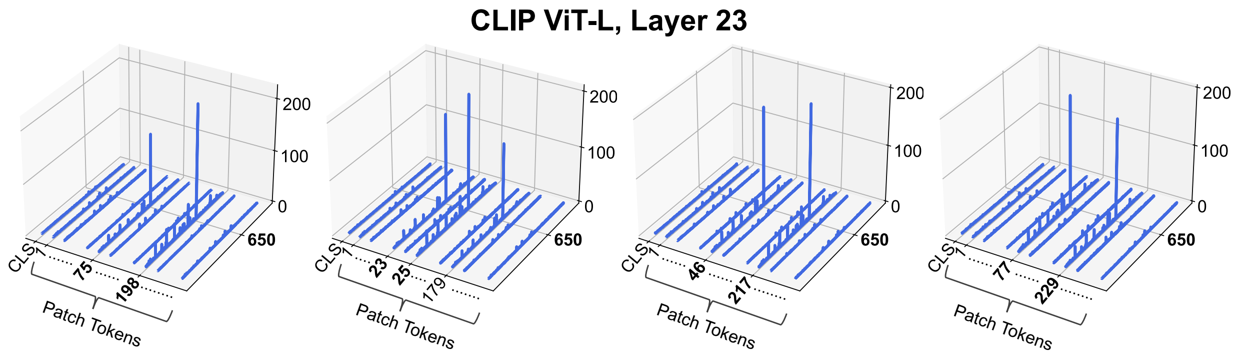


Figure 4.44: Illustration of massive activations in CLIP ViT-L for the 4 images shown in Figure 4.43.

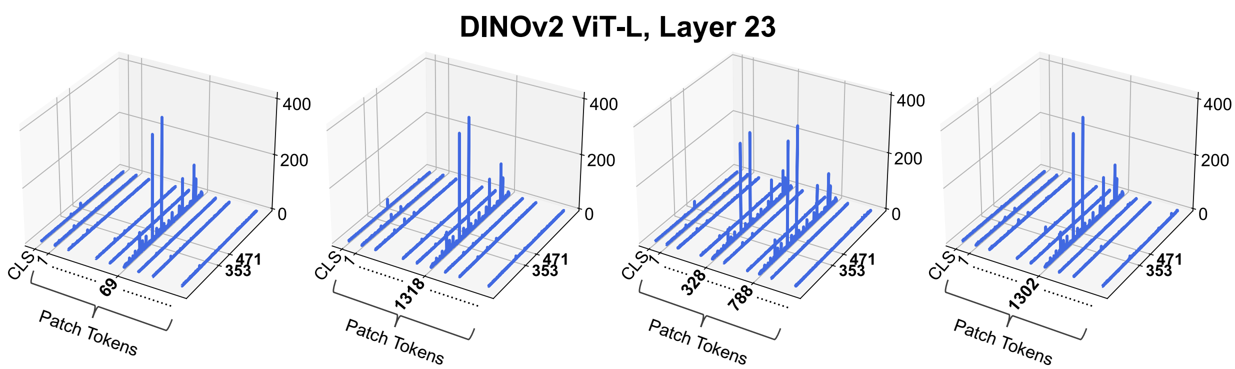


Figure 4.45: Illustration of massive activations in DINOv2 ViT-L for the 4 images shown in Figure 4.43.

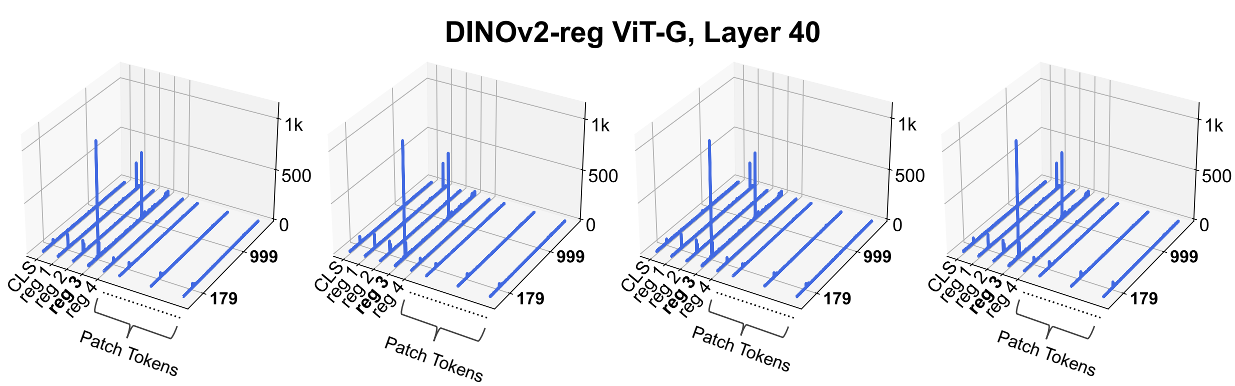


Figure 4.46: Illustration of massive activations in DINOv2 ViT-G for the 4 images shown in Figure 4.43.

4.8.10 Registers are Biases in Masked Autoencoders

In Masked Autoencoders (MAEs) [193], a dummy token is added to ViTs during pretraining. In one fine-tuning pipeline of MAEs, fine-tuning is done based on the average pooled features of all patch tokens. In these MAE models, this dummy token is equivalent to a register token. Here we maintain the register token features as constant across the output features of *all* layers in ViTs, which we denote as `Fix-Reg-Mean`. These fixed values are computed as the average register features over 10k ImageNet training images. Table 4.6 shows the results. We can see that setting register features to fixed values does not affect model performance. This result further supports our argument that registers function as biases within ViTs.

	MAE with 1 register		
ImageNet acc	ViT-B	ViT-L	ViT-H
Original	82.6	85.5	86.7
`Fix-Reg-Mean`	82.6	85.5	86.7

Table 4.6: Registers are biases in Masked Autoencoders (MAEs).

4.8.11 Layer-Level Analysis

Figure 4.47, 4.48 and 4.49 detail the layer-level analysis results for all ViTs examined in this paper (also summarized in Table 4.8). Different from LLMs, some ViTs do not exhibit massive activations, e.g., MAE ViT-B/L and DINOv2 ViT-S. For ViTs where we observe massive activations, e.g., CLIP ViT-L and DINOv2 ViT-L, the trend across layers differs from LLMs. For instance, in the case of DINOv2 ViT-L, massive activations are observed in the later stages of this model but are absent in the output of the final layer.

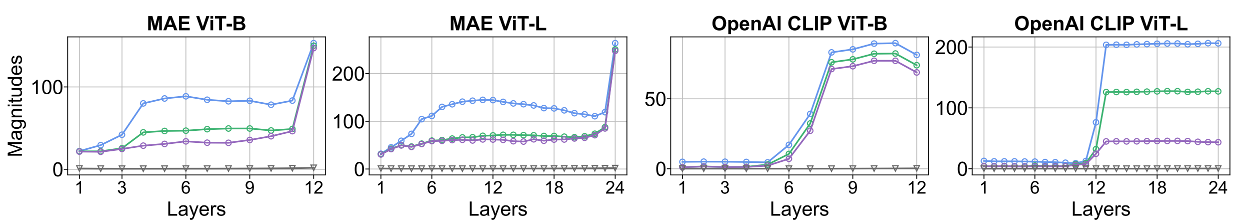


Figure 4.47: Layer-level analysis for ViTs in MAE and CLIP.

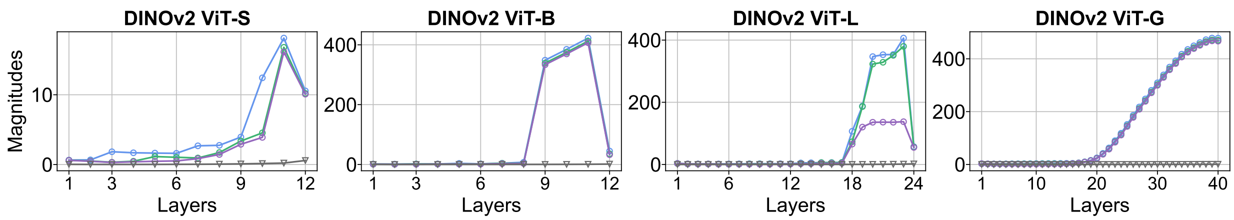


Figure 4.48: Layer-level analysis for ViTs in DINOv2.

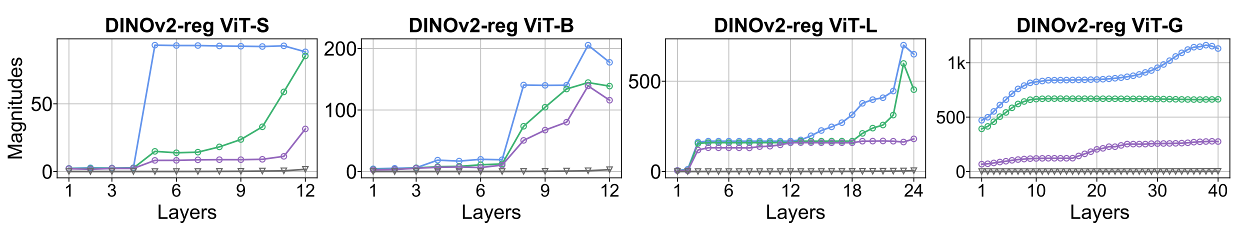


Figure 4.49: Layer-level analysis for ViTs in DINOv2-reg.

4.8.12 Models and Datasets

Table 4.7 and Table 4.8 list the information of the LLM and ViT models used in this paper.

Model family	Model name	Layers	Dimensions	Heads	Huggingface model id
LLaMA2	LLaMA2-7B	32	4096	32	meta-llama/Llama-2-7b-hf
	LLaMA2-13B	40	5120	40	meta-llama/Llama-2-13b-hf
	LLaMA2-70B	60	6656	52	meta-llama/Llama-2-70b-hf
	LLaMA2-7B-Chat	32	4096	32	meta-llama/Llama-7b-chat-hf
	LLaMA2-13B-Chat	40	5120	40	meta-llama/Llama-2-13b-chat-hf
	LLaMA2-70B-Chat	60	6656	52	meta-llama/Llama-2-70b-chat-hf
LLaMA3	LLaMA3-8B	32	4096	32	meta-llama/Meta-Llama-3-8B
	LLaMA3-70B	80	8192	64	meta-llama/Meta-Llama-3-70B
Mistral	Mistral-7B	32	4096	32	mistralai/Mistral-7B-v0.1
	Mistral-8x7B	32	4096	32	mistralai/Mistral-8x7B-v0.1
	Mistral-7B-Instruct	32	4096	32	mistralai/Mistral-7B-Instruct-v0.2
	Mistral-8x7B-Instruct	32	4096	32	mistralai/Mistral-8x7B-Instruct-v0.1
Phi	Phi-2	32	2560	32	microsoft/phi-2
MPT	MPT-7B	32	4096	32	mosaicml/mpt-7b
	MPT-30B	48	7168	64	mosaicml/mpt-30b
Falcon	Falcon-7B	32	4544	71	tiiuae/falcon-7b
	Falcon-40B	60	8192	128	tiiuae/falcon-40b
OPT	OPT-7B	32	4096	32	facebook/opt-6.7b
	OPT-13B	40	5120	40	facebook/opt-13b
	OPT-30B	48	7168	56	facebook/opt-30b
	OPT-66B	64	9216	72	facebook/opt-66b
GPT-2	GPT-2	12	768	12	gpt2
	GPT-2-Medium	24	1024	16	gpt2-medium
	GPT-2-Large	36	1280	20	gpt2-large
	GPT-2-XL	48	1600	25	gpt2-xl

Table 4.7: Relevant information of LLM models we experimented with in this work.

Model family	Model size	Layers	Dimensions	Heads	Huggingface model id
DINOv2	ViT-S	12	384	6	timm/vit_small_patch14_dinov2.lvd142m
	ViT-B	12	768	12	timm/vit_base_patch14_dinov2.lvd142m
	ViT-L	24	1024	16	timm/vit_large_patch14_dinov2.lvd142m
	ViT-G	40	1536	24	timm/vit_giant_patch14_dinov2.lvd142m
DINOv2-reg	ViT-S	12	384	6	timm/vit_small_patch14_reg4_dinov2.lvd142m
	ViT-B	12	768	12	timm/vit_base_patch14_reg4_dinov2.lvd142m
	ViT-L	24	1024	16	timm/vit_large_patch14_reg4_dinov2.lvd142m
	ViT-G	40	1536	24	timm/vit_giant_patch14_reg4_dinov2.lvd142m
MAE	ViT-B	12	768	12	timm/vit_base_patch16_224.mae
	ViT-L	24	1024	16	timm/vit_large_patch16_224.mae
	ViT-H	32	1280	16	timm/vit_huge_patch16_224.mae
CLIP	ViT-B	12	768	12	timm/vit_base_patch16_clip_224.openai
	ViT-L	24	1024	16	timm/vit_large_patch14_clip_224.openai

Table 4.8: Relevant information of ViT models we experimented with in this work.

We list the datasets used in this work and relevant license information:

- RedPajama [45]: Apache License, Version 2.0
- OpenWebText2 [44]: MIT License
- C4 [36]: Open Data Commons Attribution License 1.0 license
- PG-19 [185]: Apache License, Version 2.0
- WikiText [63]: Creative Commons BY-SA 3.0 license
- MMLU [88]: MIT License
- BoolQ [98]: Creative Commons BY-SA 3.0 license
- PIQA [188]: The license status is unclear
- WinoGrande [101]: Apache License, Version 2.0
- ARC easy and challenge [102]: Creative Commons BY 4.0 license
- ImageNet [127]: The license status is unclear

4.9 Conclusion and Discussion

Autoregressive training of large Transformers has brought significant advances in natural language processing. This study reveals the widespread existence of *massive activations* in these Large Language Models (LLMs). The values of these activations are input agnostic but crucial for model performance, despite their extremely rare quantity. We establish a close connection between massive activations and the self-attention mechanism. We show that LLMs use them to implement an implicit form of biases for attention computation. Our findings also generalize well to Vision Transformers (ViTs). We hope the new results presented in this work contribute to a deeper understanding of today's large-scale foundation models.

We discuss some practical implications and future directions of this work. First, the presence of activations with large magnitudes has been widely known as a major challenge in effectively quantizing LLMs [57, 58]. This paper identifies a new type of outlier activations in LLMs, and we hope our findings will be of value to research on LLM compression. Second, attention maps that allocate excessive attention probabilities to a few fixed tokens may be undesirable for mechanistic interpretability [186]. Our proposed attention formulation could make the resulting attention maps in LLMs more interpretable, and potentially benefit downstream applications [191]. Finally, our investigation of the new attention formulation is focused on its effects on massive activations, and our experiments were limited to a small GPT-2 model due to computational resource constraints. It would be interesting to see how our results generalize to models at larger scales, and how our attention formulation could affect the training stability [167] of modern LLMs.

Chapter 5

Idiosyncrasies in Large Language Models

5.1 Overview

In this work, we unveil and study idiosyncrasies in Large Language Models (LLMs) – unique patterns in their outputs that can be used to distinguish the models. To do so, we consider a simple classification task: given a particular text output, the objective is to predict the source LLM that generates the text. We evaluate this synthetic task across various groups of LLMs and find that simply fine-tuning existing text embedding models on LLM-generated texts yields excellent classification accuracy. Notably, we achieve 97.1% accuracy on held-out validation data in the five-way classification problem involving ChatGPT, Claude, Grok, Gemini, and DeepSeek. Our further investigation reveals that these idiosyncrasies are rooted in word-level distributions. These patterns persist even when the texts are rewritten, translated, or summarized by an external LLM, suggesting that they are also encoded in the semantic content. Additionally, we leverage LLM as judges to generate detailed, open-ended descriptions of each model's idiosyncrasies. Finally, we discuss the broader implications of our findings, including training on synthetic data, inferring model similarity, and robust evaluation of LLMs. Code is available at github.com/locuslab/llm-idiosyncrasies.

5.2 Introduction

As the adoption of generative models such as LLMs accelerates, it becomes increasingly important to understand the origin and provenance of such generated content. While a great deal of past work has focused on the classification of human-written and AI-written content [228, 243, 244, 245], there has been little work on classifying *between* content generated by different LLMs, either between the outputs of entirely different models or between those of different variants of the same model family. If possible, the ability to distinguish between source models in this manner would be valuable for a number of applications: it could shed light on the relative uptake of different LLMs, beyond what is reported by individual companies, and on the nature of data used to build different models. Additionally, it could offer insights into what features the output generation are most "unique" to each LLM.

In this paper, we investigate whether LLMs exhibit idiosyncrasies that enable their outputs to

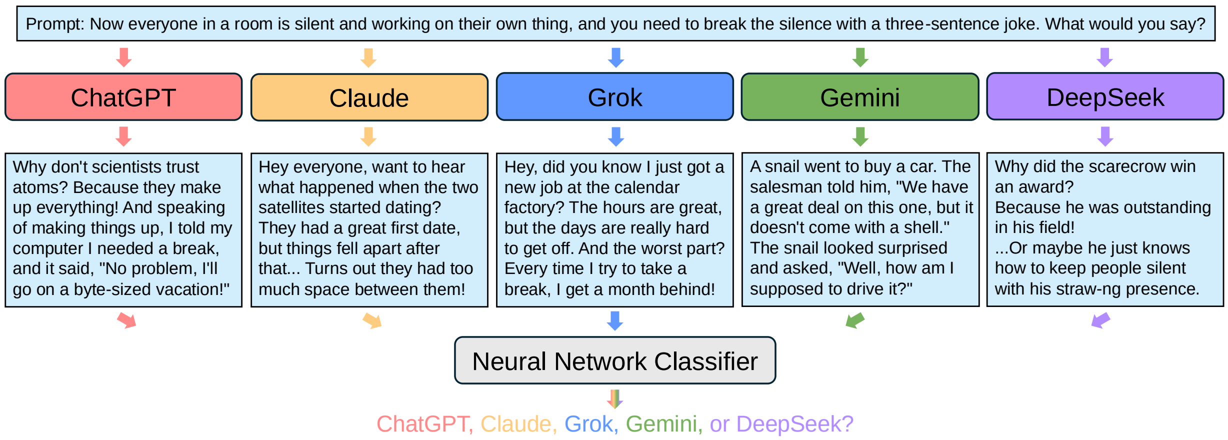


Figure 5.1: **Our framework for studying idiosyncrasies in Large Language Models (LLMs).** We show that each LLM is unique in its expression. In the example shown here on ChatGPT, Claude, Grok, Gemini, and DeepSeek, a neural network classifier is able to distinguish them with a near-perfect 97.1% accuracy.

be reliably differentiated. Drawing inspirations from recent studies on dataset bias in computer vision [236, 265], which demonstrated that images from different large-scale vision datasets can be accurately distinguished by a standard neural network classifier, we consider a similar synthetic classification task to assess the separability of responses generated between different LLMs. Specifically, we sample a large number of text outputs from each LLM using the same set of prompts and then train a classifier to recognize which model generates a specific text. Figure 5.1 provides an overview of our framework. The illustrated example on ChatGPT, Claude, Grok, Gemini, and DeepSeek presents a five-way classification problem.

We find that a classifier based upon simple fine-tuning text embedding models on LLM outputs is able to achieve remarkably high accuracy on this task. This indicates the clear presence of idiosyncrasies in LLMs. The observation is highly robust over a large variety of LLM combinations. For instance, trained on the combined set of texts from ChatGPT, Claude, Grok, Gemini, and DeepSeek, a model can achieve 97.1% classification accuracy on the *held-out* validation data, compared to a 20.0% chance-level guess. Within the same model family, we obtain a non-trivial 59.8% accuracy across 4 model sizes in Qwen-2.5 series [261]. Further, we observe strong out-of-distribution generalization of these classifiers when tested on responses from prompts outside the training distribution.

We observe several interesting properties of this task. When controlling the length and format of outputs through prompt instructions, we still obtain high classification accuracy. Furthermore, for post-trained LLMs, the classifier demonstrates non-trivial accuracy even with only the first few tokens of the generated text. However, when classifying generations from the *same* LLM but using different sampling strategies, we achieve accuracy only slightly above the chance level. In addition, we observe certain behaviors of this task that resemble those of standard text classification, where improvements in text embeddings and availability of larger training datasets lead to better classification performance.

We analyze the contributing factors to the idiosyncratic behaviors in LLMs. Our analysis is based on isolating different levels of information through text transformations. We find that after randomly shuffling words in the LLM-generated responses, we observe a minimal decline in classification accuracy. This suggests that a substantial portion of distinctive features is encoded in the word-level distribution. We then highlight distinct sets of characteristic phrases that are consistently associated with each LLM. We also find that markdown formatting contributes to a moderate degree of idiosyncrasies in LLMs following post-training.

At the same time, we obtain over 90% accuracy when the word distribution is disrupted through transformations that preserve semantics, such as rephrasing or translating. Even with the most aggressive transformation – summarizing, classification accuracy remains well above chance-level guess. This finding implies that semantic information also shapes the idiosyncrasies in LLMs. Through open-ended language analysis, we provide further insights into these characteristics. For instance, ChatGPT has a preference for detailed, in-depth explanations, whereas Claude produces more concise and direct responses, prioritizing clarity.

Last, we discuss the broader implications of our findings. One should be cautious when using synthetic data to train LLMs, as we show that many of these idiosyncrasies can be inherited in such a process. Our framework also serves as a tool for assessing model similarities among frontier models, either open-source or proprietary. In addition, we discuss how the idiosyncrasies in LLMs can be used maliciously to manipulate voting-based leaderboards, therefore highlighting the need for more robust evaluation methodologies.

5.3 Evaluating Idiosyncrasies in LLMs

Large Language Models (LLMs) share several core characteristics. The majority of them are based on the Transformer architecture [1], which we consider in this paper. Second, they are trained using an auto-regressive objective [2], where they predict the next token in a sequence based on preceding context. Lastly, their training datasets significantly overlap, often incorporating vast and diverse sources such as Common Crawl, Wikipedia and Stack Overflow. Given these similarities, it is natural to ask: do LLMs speak in the same way? If not, how can we effectively measure the degree of their differences?

To address these questions, we construct a synthetic task focused on classifying outputs from different LLMs. Consider N LLMs, denoted as $f_1, \ldots, f_N$, where each f_i takes an input prompt p and outputs a text completion o. For a given dataset $\mathcal{D}$ of prompts, the outputs produced by each LLM f_i are denoted as $\mathcal{O}_i$. We approach this problem with a straightforward setup. For N output sets $\mathcal{O}_i$, we formulate a N-way classification task, where the objective is to predict which LLM produced each output. If outputs of different LLMs were drawn from the same distribution, classification accuracy would not be better than random chance. Thus, we use the classification performance of this synthetic task as a measure of idiosyncrasies in LLMs.

Our task is formulated as a sequence classification problem, for which fine-tuning BERT-style models is a common approach [251]. In this work, we fine-tune a more recent and competitive sequence embedding model based on decoder-only Transformers: LLM2vec [207]. We attach a N-way classification head to the extracted embeddings and use LoRA-based fine-tuning to both the model weights and the linear head. Input sequences are truncated to a maximum length of 512 tokens. We evaluate the model's performance on a held-out validation set and report the

ChatGPT	Claude	Grok	Gemini	DeepSeek	acc. (chat)
✓	✓				99.3
✓		✓			97.7
✓			✓		98.7
✓				✓	97.2
	✓	✓			99.7
	✓		✓		99.6
	✓			✓	99.6
		✓	✓		99.4
		✓		✓	98.7
			✓	✓	99.9
✓	✓	✓	✓	✓	97.1

(a) chat APIs

Llama	Gemma	Qwen	Mistral	acc. (instruct)	acc. (base)
✓	✓			99.9	98.3
✓		✓		97.8	81.7
✓			✓	97.0	96.3
	✓	✓		99.9	98.3
	✓		✓	99.9	98.4
		✓	✓	96.1	95.7
✓	✓	✓	✓	96.3	87.3

(b) instruct and base LLMs

Table 5.1: **Classification accuracies for various LLM combinations.** *Top*: results for chat APIs. *Bottom*: results for instruct and base LLMs. Check marks (✓) denote the models included in each combination. We observe high classification accuracies consistently across all model combinations, indicating the presence of distinct idiosyncrasies in LLMs.

classification accuracy. Additional training details are provided in Appendix 5.7.1.

5.3.1 Main Observations

We observe surprisingly high accuracies by neural networks to classify LLM outputs. This observation is robust across different settings, *e.g.*, across model families and sizes.

We describe the LLMs we use to generate the output datasets $\mathcal{O}_{1,\cdots,N}$. For a comprehensive and fair comparison across model families, we categorize three groups of LLMs:

1. Chat APIs ("chat"): This category includes state-of-the-art LLMs that are primarily accessible via APIs. We consider GPT-4o [13], Claude-3.5-Sonnet [15], Grok-2 [16], Gemini-1.5-Pro [263], and DeepSeek-V3 [6]. For simplicity, we refer to them as ChatGPT, Claude, Grok, Gemini and DeepSeek. Their architectures and weights remain proprietary and undisclosed, with the exception of DeepSeek.
2. Instruct LLMs ("instruct"): These models are trained to generate high-quality responses from human instructions. We consider four LLMs of similar sizes across different families: Llama3.1-8b [25], Gemma2-9b [198], Qwen2.5-7b [261] and Mistral-v3-7b [173]. We will refer to them as Llama, Gemma, Qwen and Mistral.
3. Base LLMs ("base"): These are base versions of instruct LLMs. They are obtained by pretraining on extensive text corpora without any post-training stage.

Throughout the paper, we refer to these three categories as "chat", "instruct", and "base" respectively. For a given prompt dataset, we collect 11K text sequences, splitting them into 10K for training and 1K for held-out validation. The same split is used across all LLMs. For chat APIs and instruct LLMs, we generate outputs from UltraChat [225], a diverse dialogue and instruction dataset. For base LLMs, we synthesize new texts using prompts from FineWeb [223], a high-quality LLM pretraining dataset. More details on response generation are in Appendix 5.7.2.

Across model families. In Table 5.1, we report the results for classifying outputs from various combinations of chat APIs (Table 5.1a) and instruct / base LLMs (Table 5.1b). In each of the three LLM groups, we enumerate all (C_N^2) possible pairwise combinations when choosing 2 out of N models in the top panel of each table, as well as the case including N models in the bottom row. For the binary classification task, the neural network consistently achieves over 90% accuracy, with only one exception. Notably, for chat APIs and instruct LLMs, many combinations reach as high as 99% accuracy. In the more challenging N-way classification tasks, our classifiers maintain strong performance, achieving at least 87.3% accuracy across three groups. These results highlight the idiosyncrasies across different LLMs. We refer readers to Chapter 5.7.5 for the confusion matrices of our classifiers.

Within the same model family. We evaluate sequence classification performance when distinguishing responses from LLMs within the same model family. Note that models from the same family typically share common training procedures, *e.g.*, pretraining datasets and optimization schedule. First, we analyze the impact of model size by considering four Qwen2.5 instruct LLMs with 7B, 14B, 32B, and 72B parameters. As shown in Table 5.2, the classification task becomes

7b	14b	32b	72b	instruct
✓	✓			77.0
✓		✓		81.2
✓			✓	83.4
	✓	✓		63.1
	✓		✓	85.5
		✓	✓	84.8
✓	✓	✓	✓	59.8

Table 5.2: **Classification within Qwen2.5 model family.** The classifier can differentiate responses between LLMs within the same model family with reasonably well accuracies.

train / test	UltraChat	Cosmopedia	LmsysChat	WildChat
UltraChat	96.3	98.9	89.9	92.4
Cosmopedia	95.7	99.8	88.3	94.9
LmsysChat	94.7	97.2	91.8	92.0
WildChat	95.1	99.1	90.2	95.7

Table 5.3: **Robust generalization to out-of-distribution responses.** We train classifiers on LLM outputs from one prompt dataset and tested on those from another.

more difficult, but our classifiers remain reasonably well above chance accuracy when distinguishing LLMs within the same family. In the binary classification setup, the highest accuracy reaches 85.5%, whereas in the full combination setup, the accuracy becomes 59.8%. In addition, we observe high accuracies when classifying responses from base and instruct versions of the same model. For example, our classifiers achieve 96.8% accuracy when distinguishing outputs from Qwen2.5-7b base and instruct models.

Generalization to out-of-distribution responses. We find that our classifiers generalize robustly to responses beyond their training distribution. To evaluate this, we collect responses from instruct LLMs across four diverse datasets: *i.e.*, UltraChat, Cosmopedia [274], LmsysChat [275], and WildChat [278]. These datasets originate from different sources and are designed for various purposes – Cosmopedia is designed for synthetic data generation, LmsysChat and WildChat capture real-world user interactions, while UltraChat consists primarily of synthetic responses. For each dataset, we train a classifier on a group of model responses and evaluate the classifier across all four datasets. We use instruct LLMs for this experiment. As shown in Table 5.3, our classifiers generalize well across different datasets, indicating that they learn very robust and transferable patterns.

5.3.2 Controlled Experiments

We analyze the behaviors of the synthetic classification task in several controlled settings. *From now on, we only report the accuracy of the N-way classification task in each group.*

	original	length control	format control
instruct LLMs	96.3	93.0	91.4

Table 5.4: **Controlling LLM outputs with prompts.** An instruction is added to the original prompt to specify the output length and format. *Length control* limits responses to one paragraph. *Format control* ensures that responses are in plain text without any format.

Prompt-level interventions. We assess the degree of idiosyncrasies in LLM outputs with explicit prompt-level interventions. Specifically, we modify the original prompt by incorporating additional instructions to constrain response length and format. We then perform sequence classification on the resulting outputs. Our interventions are:

- Length control: *Please provide a concise response in a single paragraph, limited to a maximum of 100 words.*
- Format control: *Please provide your response in plain text only, avoiding the use of italicized or bold text, lists, markdown, or HTML formatting.*

LLM outputs after these interventions are presented in Chapter 5.7.10. We find that LLMs can follow the additional instructions in generating responses.

Table 5.4 presents the results for this analysis. We can see that neural networks still perform excellently for classifying LLM outputs applied with length and format control prompts. These findings suggest that LLM characteristics are deeply embedded in the generated text, persisting despite surface-level constraints on length and formatting.

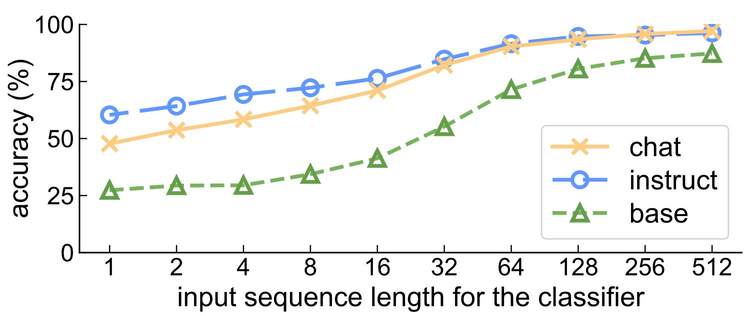


Figure 5.2: **Ablations on input length of text embedding models.** Classification accuracies improve as the text embedding models capture more context. Performance begins to saturate beyond an input sequence length of 256. Note that the three lines represent different groups of LLMs and are not directly comparable.

Input length of text embedding models. We control the number of input tokens to the text embedding models. Specifically, we truncate each response to a fixed number of tokens in a left-to-right fashion. Figure 5.2 presents the results. Across three groups of LLMs, the classification task benefits from seeing an increased number of tokens. Intriguingly, for chat APIs and instruct LLMs, we observe around 50% accuracy with just a single text token. This suggests that the initial token in a response contains certain distinctive signals for the classification problem. In Section 5.4.1, we provide further evidence supporting this observation.

	greedy	softmax	top-k	top-p
greedy	-	-	-	-
softmax	59.6	-	-	-
top-k	58.2	50.0	-	-
top-p	52.9	51.0	52.1	-

Table 5.5: **Classifications with different sampling methods.** Distinguishing responses generated by the same model using different sampling strategies is only marginally better than chance accuracy. The results are on Llama3.1-8b instruct model's responses.

Sampling methods. We consider outputs when sampled using different decoding strategies. Specifically, we use four widely used sampling methods: greedy decoding, temperature softmax, top-k, and top-p sampling. For each method, we generate a set of responses from the LLM. We then fine-tune the LLM2vec embedding model to predict the sampling method responsible for each response.

Table 5.5 presents the results for all pairs of sampling methods. Notably, the accuracy of distinguishing between responses generated by the same LLM remains relatively low, with the highest accuracy across all configurations being 59%. Furthermore, in a more fine-grained 5-way classification task distinguishing softmax sampling at five different temperatures ($T = 0$, 0.25, 0.5, 0.75, 1), we obtain an accuracy of 37.9%, only marginally better than the random chance level of 20%. These results suggest that outputs from the same LLM are not easily separable based on decoding strategies.

Text embedding models. We vary the underlying pretrained embedding models for sequence classification. The default setting we used in previous parts is fine-tuning the LLM2vec embedding models. We consider various generations of embeddings models spanning across architectures and training methods: ELMo [269], BERT [18], T5 [208], GPT-2 [2], and LLM2vec [207]. Details on the fine-tuning setting can be found in Chapter 5.7.3.

method	chat	instruct	base
ELMo	90.8	91.0	69.8
BERT	91.1	91.5	66.0
T5	90.5	89.8	67.9
GPT-2	92.1	92.3	80.2
LLM2vec	**97.1**	**96.3**	**87.3**

Table 5.6: **Different sequence embedding models.** LLM2vec achieves the best performance in classifying outputs from various LLMs among the five embedding models we study.

Table 5.6 shows the results. All sequence embedding models can achieve very high accuracies. The classification performance improves with more advanced sequence embedding models. Among all methods, LLM2vec demonstrates the best performance, achieving 97.1% on chat APIs, 96.3% on instruct LLMs, and 87.3% on base LLMs.

Training data size. We vary the number of training samples generated by LLMs and train the classifier with the same total number of iterations. We present the results in Figure 5.3. The performance of the classifier increases as it is trained with more training samples. This trend is consistently observed across chat APIs, instruct LLMs, and base LLMs. Furthermore, as few as 10 training samples, the classifier achieves non-trivial accuracy (*e.g.*, 40.3% on instruct LLMs), surpassing 20% chance-level guess.

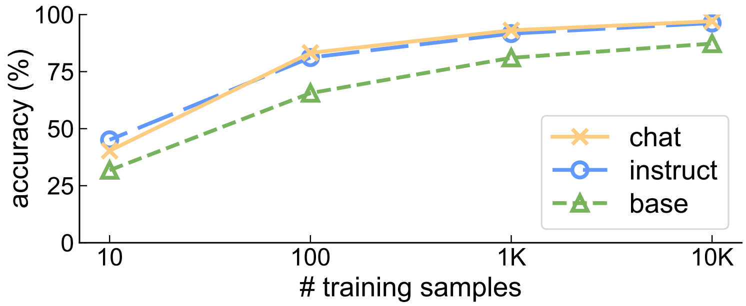


Figure 5.3: **Different numbers of training samples.** Our sequence classifiers benefit from more training samples. The classification performance converges when using about 10K training samples.

5.4 Concrete Idiosyncrasies in LLMs

We have shown that modern neural networks can achieve excellent accuracies in classifying which LLM generates a given response. Here we leverage classical text similarity metrics – ROUGE-1 [266], ROUGE-L [266], and BERTScore [18] – to quantify lexical differences between LLM outputs. We compute the mean F1-score for each metric across all response pairs generated by any two different chat API models given the same prompt. For comparison, we also measure the similarity between responses sampled within the same model. As shown in Table 5.7, responses from different LLMs exhibit lower text similarities than those from the same model.

	across LLMs	within an LLM
ROUGE-1	0.499	0.660
ROUGE-L	0.256	0.414
BERTScore*	0.220	0.482

Table 5.7: **Text similarity scores.** We evaluate the text similarity of LLM outputs using ROUGE-1, ROUGE-L, and BERTScore. * We follow [267] to rescale BERTScore with respect to the human baseline. The results indicate that responses from different LLMs exhibit low lexical similarity.

In the following, we identify concrete idiosyncrasies in LLMs across three dimensions: words and letters, markdown formatting elements, and semantic meaning. For each dimension, we apply text transformations to isolate potential idiosyncrasies and assess their impacts on classification performance. We then highlight specific patterns within each dimension that distinguish LLMs.

ChatGPT	Claude
Our products feature innovative sustainable materials, such as …	According to the text, Kai Fusser believes that traditional cardio …
Certainly! If you're looking for cheese alternatives to replace Brie in your …	Based on the text provided, here are the key details about Armon Binns' …
Overall, while there are challenges, Tanzania is making progress …	While many winter sports in the Pyrenees are similar to those found …
Sure! Here's a simple guide to cooking a juicy salmon fillet: …	This appears to be a fragment of poetry that creates a pastoral …

(a) characteristic phrases

ChatGPT	Claude
1. **Deliver Exceptional Service**: The foundation of word-of-mouth marketing is consistent excellence. Providing top-notch services or …	1. Deliver Exceptional Service • Consistently exceed customer expectations • Focus on quality and attention …
Ingredients: • 2 (3 oz) packages of orange-flavored Jello • 1 cup tonic water (this is what …	Ingredients: • 2 boxes orange-flavored Jello • 1 can evaporated milk • Tonic water …

(b) unique markdown formatting

Figure 5.4: **Example responses from ChatGPT and Claude, showcasing their idiosyncrasies**: characteristic phrases (*left*) and unique markdown formatting (*right*). For clarity, we highlight each characteristic phrase with underline and model-specific color.

5.4.1 Words and Letters

Text shuffling. To decouple the effects of words and letters from other factors, we remove special characters in LLM-generated responses, such as punctuations, markdown elements, and excessive white spaces. This ensures that each response consists solely of words separated by a white space. Additionally, we apply two shuffling strategies to the preprocessed text: word-level and letter-level shuffling. These transformations disrupt the natural order and force the classifier to learn patterns from raw text statistics. Table 5.8 presents the classification results.

Classifiers trained on responses without special characters achieve accuracies close to those using the original responses, *i.e.*, 95.1% for chat APIs, 93.8% for instruct LLMs, and 75.4% for base LLMs. Likewise, using word-shuffled responses yields high accuracies comparable to the original ones. Further, we plot the frequencies of several commonly used words from five chat APIs in Figure 5.5 (*left*). We observe distinct patterns among models, even for frequent English words: Claude has much lower frequencies for words like "the", "and", "to", and "of" than other chat APIs. These results suggest that *special characters and word order are not essential for distinguishing LLMs*; *word choices reflect substantial idiosyncrasies across models.*

In contrast, shuffling at the letter level results in a substantial drop in accuracy (49%-56%), approaching chance-level performance. This indicates that letter-level statistics alone are not sufficient for predicting LLM identities. To qualitatively visualize distinctions in letter distributions

	chat	instruct	base
original	97.1	96.3	87.3
removing special characters	95.1	93.8	75.4
shuffling words	88.9	88.9	68.3
shuffling letters	39.1	38.6	38.9

Table 5.8: **Classifications with only words and letters.** While removing special characters and shuffling words have little impact on accuracies, shuffling letters greatly reduces the performance.

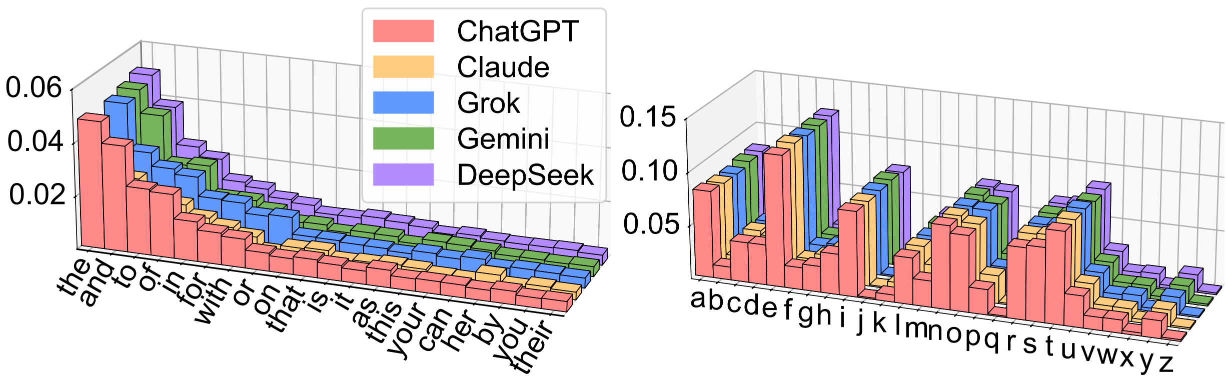


Figure 5.5: **Frequencies of words and letters.** The top 20 most frequently used words of LLMs (left) exhibit distinct patterns for each model, but their letter frequencies (right) are very similar. Results are on the chat API models.

across models, Figure 5.5 (*right*) shows the frequency distribution of letters in responses generated by chat APIs. Different LLMs share almost identical letter distributions, indicating that *letters contribute minimally to idiosyncrasies in LLMs.*

Characteristic phrases. We use Term Frequency-Inverse Document Frequency (TF-IDF) to highlight characteristic phrases inside LLM-generated responses that reflect each model's word choices. Formally, we treat each LLM response as a document and then extract TF-IDF features on all uni-gram and bi-gram words. We then train a N-way logistic regression model to predict the origin of responses on the extracted features. This simple linear classifier achieves 85.5% / 83.7% accuracy on chat APIs / instruct LLMs, close to 95.1% / 93.8% achieved with fine-tuning embedding models on responses without special characters (Table 5.8).

Since the coefficients of a logistic regression model provide a natural ranking for its features, we leverage these coefficients to highlight important phrases in the classification task. Figure 5.6 presents the top 10 phrases with the largest logistic regression coefficients for each of the five chat API models. Notably, these phrases often serve as transitions or emphasis in sentences. For example, ChatGPT likes to generate "such as", "certainly", and "overall", whereas Claude prefers "here", "according to", and "based on".

Figure 5.4a illustrates these characteristic phrases with example responses from ChatGPT and Claude. While ChatGPT begins responses with "certainly" and "below is", Claude usually references the original prompt using the phrases like "according to the text" and "based on the

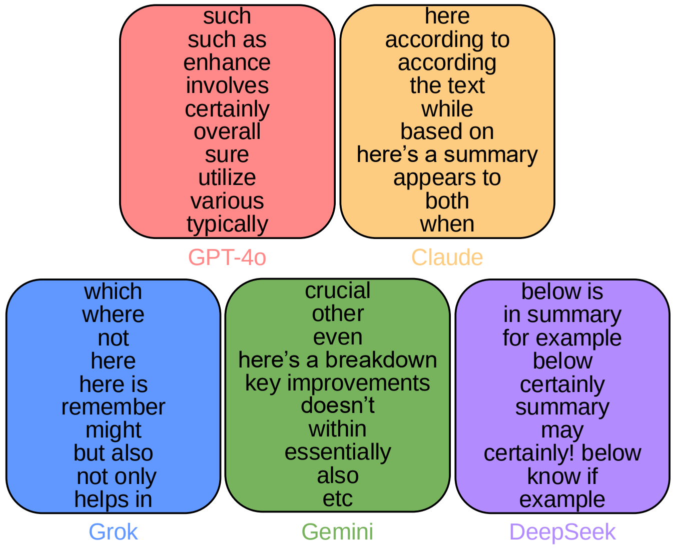


Figure 5.6: **Characteristic phrases.** We train a logistic regression model on TF-IDF features of chat APIs' outputs and extract the top 10 phrases for each LLM based on the coefficients of these features. We remove common words shared across these LLMs.

text". Moreover, Figure 5.7 reveals noticeable differences in the distribution of first word choices among chat APIs. Chapter 5.7.7 provides characteristic phrases for other LLMs.

5.4.2 Markdown Formatting

We seek to understand how each LLM formats their responses, particularly in markdown. To this end, we focus on common markdown elements used by LLMs: (1) bold text, (2) italic text, (3) header, (4) enumeration, (5) bullet point, (6) code block. We transform the LLM outputs by retaining only these formatting components while replacing other text with the marker "xxx". Chapter 5.7.10 provides examples of the transformed outputs. Table 5.9 shows the classification results after this transformation.

	chat	instruct	base
original	97.1	96.3	87.3
markdown elements only	73.1	77.7	38.5

Table 5.9: **Classifications with only markdown elements.** Using markdown elements can achieve high accuracies for chat APIs and instruct LLMs, but marginally better results for base LLMs.

Surprisingly, we observe our classifiers achieve high accuracies of 73.1% for chat APIs and 77.7% for instruct LLMs. However, the classification accuracies with base LLMs' responses are near chance-level guess (25%). This is likely because base LLMs tend to generate responses in plain text.

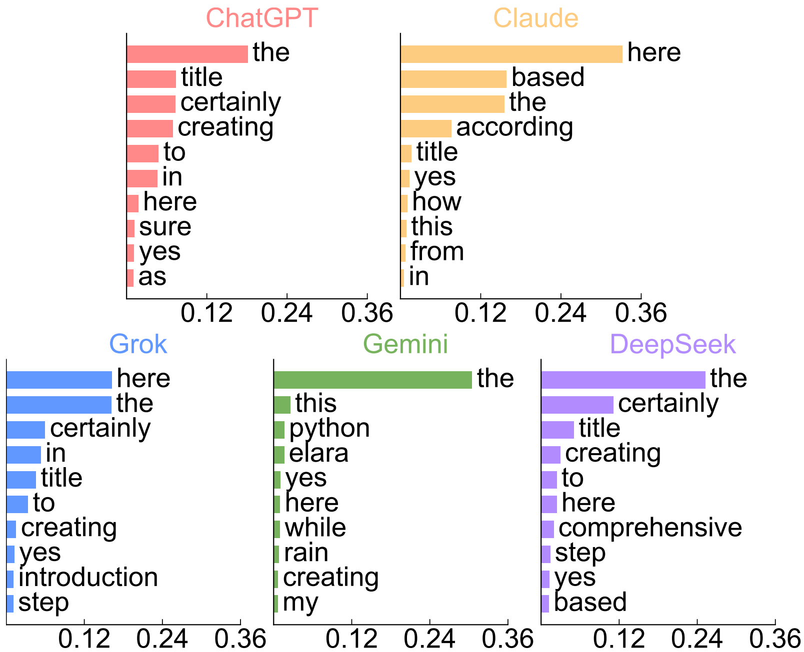


Figure 5.7: **First word.** We analyze the distribution of the first word in chat APIs' responses, with the top 10 most frequent words for each model. These differences in first-word usage explain the non-trivial accuracy with only the first word in Figure 5.2.

We count the occurrence of a markdown formatting element in each response. We then plot the distribution of these counts over all responses in Figure 5.8. Each model exhibits a unique way to format its responses. For instance, Claude (yellow) has a high density at zero in the bold text and header count distributions, indicating that it generates many responses without bold texts or headers. On the contrary, other LLMs exhibit lower values at zero and thus decorate text with these formatting elements more often.

Figure 5.4b visualizes how ChatGPT and Claude structure their responses in markdown. Interestingly, ChatGPT tends to emphasize each key point within enumerations in bold and highlight a title with markdown headers, but Claude formats text with simple enumeration and bullet points. More analysis on these markdown formatting elements for other models can be found in Chapter 5.7.8.

5.4.3 Semantics

Rewriting. One potential reason for the high classification accuracy is the unique writing style (*e.g.*, word choice, sentence structure) of each LLM. To isolate this factor, we leverage another LLM (*e.g.*, GPT-4o mini) to rewrite LLM responses. Our rewriting approaches include (see Chapter 5.7.10 for example responses after rewriting):

- Paraphrasing: *Paraphrase the above text while maintaining the semantic meaning of the original text.*
- Translating: *Translate the above text into Chinese.*
- Summarizing: *Summarize the above text in one paragraph.*

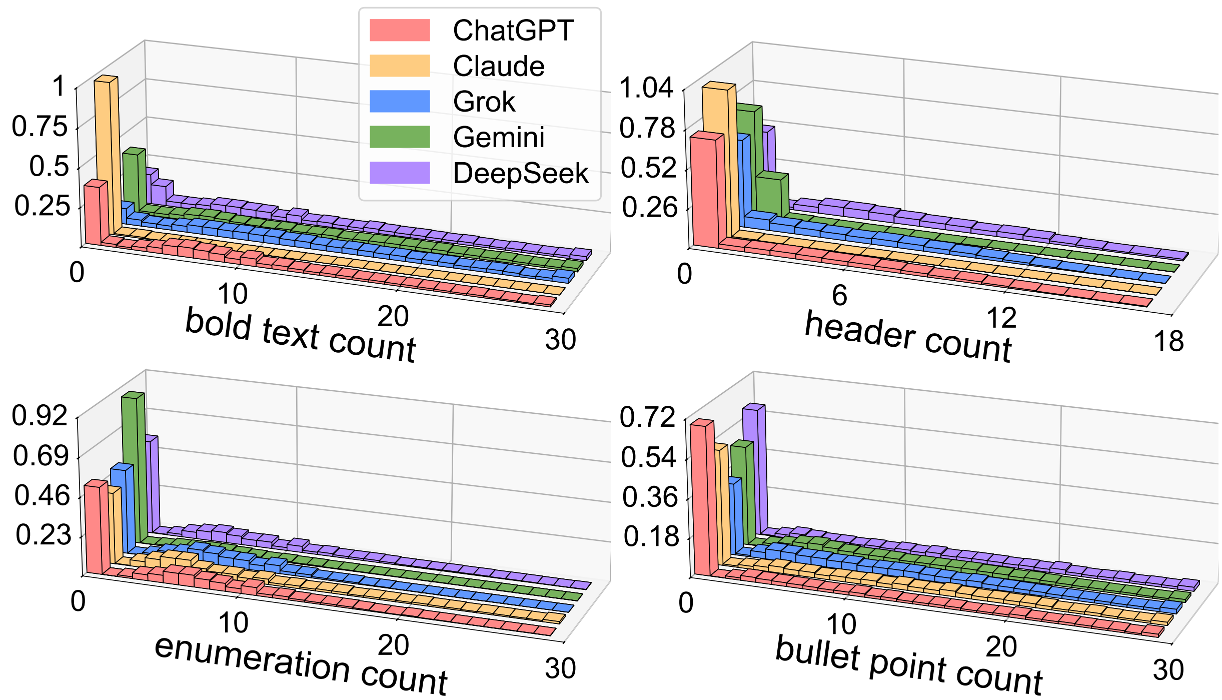


Figure 5.8: **Markdown formatting elements.** Each LLM has a distinctive distribution of markdown formatting elements.

We show the results in Table 5.10. The classifiers trained on paraphrased LLM responses maintain similar accuracy levels to those using original responses. Likewise, when using translated text, the classifiers are also able to differentiate between LLMs. These findings suggest that *the semantic meanings of words play a more significant role in predicting LLM origins than the exact word choice.*

	chat	instruct	base
original	97.8	96.3	87.3
paraphrasing	91.4	92.2	71.7
translating	91.8	92.7	74.0
summarizing	58.1	57.5	44.7

Table 5.10: **Classifications on rewritten responses.** Paraphrasing or translating LLM outputs achieves an accuracy comparable to that using original counterparts. However, summarizing these texts makes the model less capable of predicting LLM identities.

Moreover, despite a noticeable accuracy drop (*i.e.*, >38%) with the summarized text, the resulting performance remains well above chance-level guess. This remarkable ability to classify the summarized texts shows the *high-level semantic difference in LLM-generated responses.*

1. **Descriptive and Detailed Tone:** Often uses narrative styles with an informative, engaging, or vivid tone.
2. **Specific and Technical Word:** Employs descriptive and technical vocabulary, enhancing depth and specificity.
3. **Structured and Contextual Opening Lines:** Typically begins with context-setting or narrative introductions.
4. **Markdown Formatting for Organization:** Utilizes various markdown elements like headings, lists, and bold text for clarity.
5. **Comprehensive and In-Depth Content:** Offers rich detail, focusing on explanations, background, and broader topics.

ChatGPT

1. **Concise and Straightforward Tone:** Generally adopts a more direct, factual, or succinct tone.
2. **Functional and Clear Word Choices:** Prefers simple or action-oriented language prioritizing clarity and practicality.
3. **Immediate and Direct Opening Lines:** Often starts with a straightforward statement or summary without extended context.
4. **Minimal Markdown or List Use:** Relies on plain lists or simple formatting for quick reference.
5. **Focused and Summarized Content:** Concentrates on essential points and specific phenomena, avoiding extensive detail.

Claude

Figure 5.9: **Results of our open-ended language analysis on ChatGPT and Claude. ChatGPT** features descriptive language, sophisticated markdown formatting, and in-depth details, while Claude highlights straightforward tone, minimal structure, and summarized content.

Open-ended language analysis. In this part, we focus on studying the semantic difference in responses generated by LLMs. We employ another LLM (*e.g.*, ChatGPT) as a judge to provide open-ended, descriptive characterizations for each LLM's outputs. The results with other LLM judges for our language analysis are available in Chapter 5.7.9.

Specifically, we present an LLM judge with two responses – generated by different models based on the same prompt – and ask it to analyze these responses from different angles (*e.g.*, tone and content). This process is repeated multiple times to gather a comprehensive collection of analyses. Finally, we query the LLM judge to summarize these analyses into bullet points that capture the characteristics of each model. The prompts are detailed in Chapter 5.7.4.

The results of open-ended language analysis on ChatGPT *vs*. Claude are shown in Figure 5.9. For a detailed pairwise comparison of the responses, see Figure 5.16 in Chapter 5.7.10. ChatGPT is characterized by descriptive and detailed responses in an engaging tone. In contrast, Claude prioritizes simplicity with only key points and straightforward language. Additional results on chat API models and instruct LLMs are provided in Chapter 5.7.9.

5.5 Implications

In this section, we explore the broader implications of our framework, regarding synthetic data and model similarity.

Idiosyncrasies via synthetic data. Using synthetic data has become a common practice when training frontier LLMs [242, 258, 273]. We conduct supervised fine-tuning (SFT) on two base LLMs (Llama3.1-8b and Gemma2-9b) using Ultrachat, i.e., dialogues generated by ChatGPT. After the SFT stage, we train a classifier to distinguish between responses from two fine-tuned models. We find that SFT on the same synthetic dataset significantly reduces the classification accuracy from 96.5% to 59.8%, narrowing down the differences between these two models.

In addition, we generate responses from Llama3.1-8B and Gemma2-9B in instruct LLMs using UltraChat prompts. Then we fine-tune Qwen2.5-7B base LLM on each set of responses respectively. Interestingly, responses from the two resulting fine-tuned models can be classified with 98.9% accuracy, suggesting that each fine-tuned model retains the unique characteristics in its SFT data. These findings suggest that training with synthetic data can propagate the idiosyncrasies in the source model.

Inferring model similarity. Our framework offers a quantitative approach for assessing similarities between proprietary and open-weight LLMs. Given a set of N LLMs, we omit one model and train a classifier on responses from the remaining $N - 1$ models. We then evaluate which LLM the classifier associates the responses of the excluded model with. The model that is most frequently predicted as the source is considered the closest match to the excluded LLM. This process is repeated for each of the N models. For this analysis, we include the open-weight Phi-4 [242] alongside 5 chat API models. Notably, Phi-4 uses a substantial amount of synthetic data in its training.

Results are shown in Figure 5.10. Intriguingly, for Claude, Grok, and Gemini, we observe a strong tendency for their outputs to be classified as ChatGPT. For instance, when Grok is the excluded model, 82.8% of its responses are classified as ChatGPT. In addition, responses from ChatGPT and DeepSeek are frequently identified as coming from Phi-4, with 55.9% and 76.0% of their responses respectively. In turn, most of Phi-4's outputs are classified as originating from ChatGPT or DeepSeek.

Robust evaluation of LLMs. Our findings reveal a potential vulnerability in widely used LLM evaluation methodologies. It has become a common strategy to incorporate human judgement in evaluating LLMs, for instance, Chatbot Arena [276]. It is a voting-based leaderboard where users submit preferences of the responses from two random models. This benchmark has gained significant traction and is now a key reference point for frontier model development. However, exploiting the idiosyncratic property of LLM outputs, a malicious attacker can identify the model behind the candidate responses and consistently vote for the target model, thereby manipulating the leaderboard rankings. Concurrent work by [277] has demonstrated the feasibility of this attack in simulation. We hope our work brings attention to potential weaknesses in current evaluation pipelines, as they can misguide model development and optimization efforts [28].

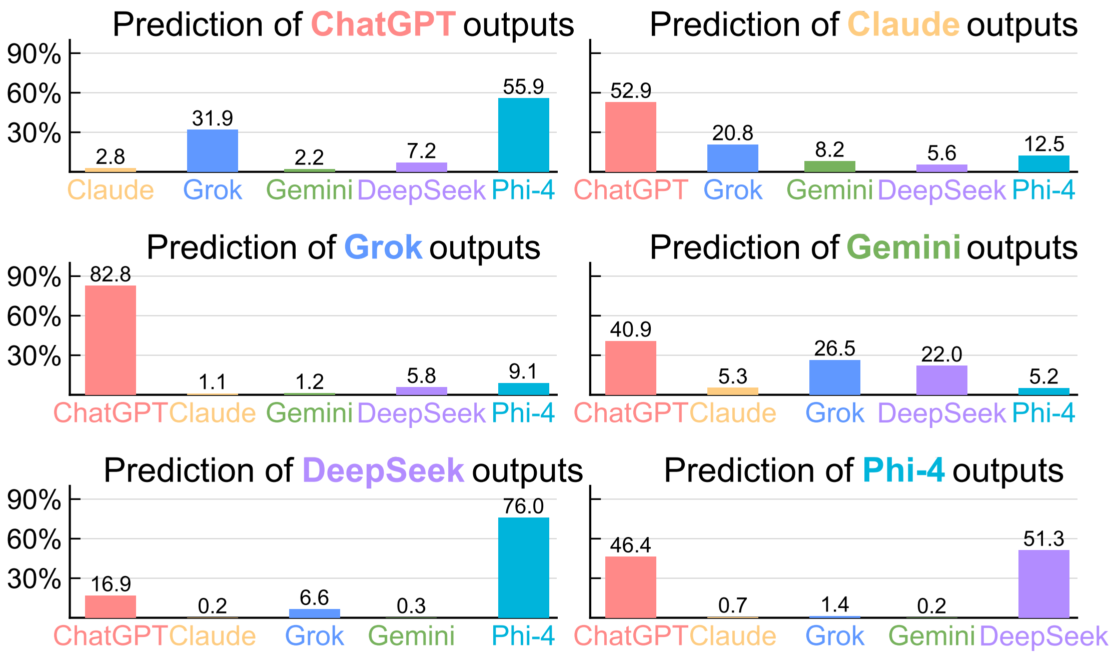


Figure 5.10: **Inferring model similarity.** We consider 6 LLMs, including 5 chat API models and Phi-4. In each subfigure, we evaluate a five-way classifier on outputs from the excluded LLM and present the distribution of predicted model origins. There is a strong tendency for LLM outputs to be predicted as ChatGPT.

5.6 Related Work

Dataset classification. [238] introduced the "Name That Dataset" experiment a decade ago to highlight the bias present in visual datasets of that time. Recently, [265] revisited this problem (termed dataset classification) and found that current large-scale, supposedly more diverse visual datasets are still very biased. [236] further identified structural and semantic components in images as key contributors to these biases. [232] and [237] applied the dataset classification framework to study bias in synthetic images and LLM pretraining datasets respectively. While the synthetic task shown in Figure 5.1 is conceptually similar to dataset classification, we focus not on training datasets but on the distinctive characteristics inherent to LLMs.

Human *vs*. machine-generated texts. Many prior works have studied the problem of determining if a text is authored by a human or an AI system [228, 229]. Model-free approaches typically use linguistic properties such as n-gram frequencies [230, 240], entropy [241, 246] or negative probability curvature [228, 247]. Other works leverage neural network features to perform this task, such as fine-tuning BERT models [248, 249]. Neural authorship attribution [253, 254] seeks not only to identify machine-generated text but also to attribute it to specific text generators. In this work, we focus on the distinguishability between LLMs rather than between AI *vs*. human.

Understanding differences between distributions. A line of research [224, 231] has used foundation models to describe qualitative differences between pairs of data distributions (*e.g.*, image datasets). [234] conducted hypothesis testing on sets of model outputs to check whether the underlying LLMs were identical. The most relevant work to us is [235], which proposed VibeCheck to understand user-aligned traits in LLM outputs. They found that LLMs often vary in styles, such as being more formal or friendly. In contrast, our work aims to identify broader generalizable patterns to interpret the high classification performance.

5.7 Additional Results

5.7.1 Implementation Details

5.7.2 Response Generation

We report our procedure for generating responses from chat APIs, instruct LLMs, and base LLMs. For chat APIs, we access a stable version of each model, including GPT-4o-2024-08-06, Claude-3.5-Sonnet-20241022, Grok-Beta, Gemini-1.5-Pro-002, and DeepSeek-Chat, through its official API between November 28, 2024, and February 6, 2025, generating responses with their default sampling setting. For instruct LLMs, we use greedy decoding to sample outputs. For base LLMs, we set the temperature to $T = 0.6$ and apply a repetition penalty of 1.1 to avoid repetitive completions.

5.7.3 Training Setup

In this part, we describe our fine-tuning process using the text embedding models on LLM responses. We use the first 512 tokens of each generated response for training and evaluation. To perform sequence classification, we add a linear layer as the classification head on top of each text embedding model. For ELMo, BERT, LLM2vec, this layer is applied to the average embeddings over all tokens in a sequence. For T5 and GPT-2, we follow the original setups [2, 208] and apply the head on the output of the last token.

For smaller text embedding models, such as ELMo, BERT, T5, and GPT-2, we fine-tune the entire model along with the classification head, searching over base learning rates $\{$3e-3, 1e-3, 3e-4, 1e-4, 3e-5, 1e-5, 3e-6, 1e-6$\}$. For the largest LLM2vec model, we employ the parameter-efficient LoRA [134] fine-tuning method with a rank of 16, LoRA α of 32, a dropout rate of 0.05, and a base learning rate of 5e-5. Table 5.11 details our basic training recipe.

5.7.4 Prompts for Open-ended Language Analysis

We detail the procedures of our open-ended language analysis in Section 5.4.3. Given the same input, we sample a pair of responses from two LLMs and present them, along with an analysis prompt (see Figure 5.11a), to an LLM judge for comparison. To avoid the LLM judge exploiting any prior knowledge of the models, we anonymize model identities using an index distribution. This process is repeated for 35 response pairs, yielding a set of detailed analyses. Finally, we use the summarization prompt (see Figure 5.11b) to distill these analyses into 5 bullet points that characterize the idiosyncrasies of each model.

config	value
optimizer	AdamW
weight decay	0.001
optimizer momentum	$\beta_1, \beta_2 = 0.9, 0.999$
training epochs	3
batch size	8
learning rate schedule	cosine decay
warmup schedule	linear
warmup ratio	10%
gradient clip	0.3

Table 5.11: Our fine-tuning recipe.

> Here are some samples from 2 different distributions of text.
>
> Sample from distribution {1, 2}: {text sample 1}
> Sample from distribution {1, 2}: {text sample 2}
>
> Determine the unique characteristics of the 2 distributions, and summarize them with five bullet points for each. Each bullet point should analyze the response from the same angle and can be easily compared by a human. Focus on key aspects that differentiate the distributions. Focus on the overall tone, specific word choices, opening lines, markdown formatting, and content. Output only 5 bullet points per distribution, no additional text.

(a) analysis prompt

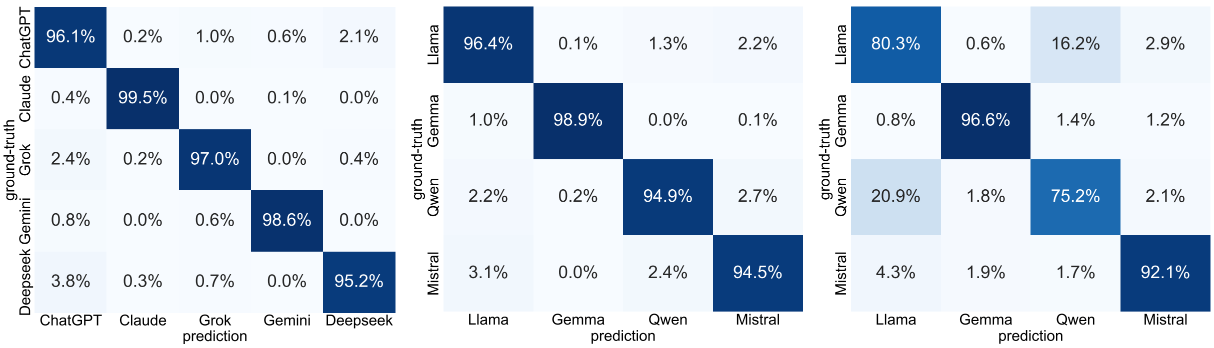
> Below are 35 summaries that compare the unique features in 2 text distributions point by point.
>
> {Analysis 1}
> {Analysis 2}
> ...
> {Analysis 35}
>
> Condense all 35 summaries into a single summary. Focus on how each summary evaluates the overall tone, specific word choices, opening lines, markdown formatting, and content. Provide exactly five bullet points for each distribution, one for each feature. Each bullet point should consist of a descriptive title characterizing the feature and a short sentence explaining it concisely. Emphasize recurring and significant features, remove the redundant ones, and ensure the titles and sentences clearly differentiate the distributions for easy human comparison. Output only 5 bullet points per distribution, no additional text.

(b) summarization prompt

Figure 5.11: Prompts in our open-ended language analysis.

5.7.5 Confusion Matrix

In Figure 5.12, we present the confusion matrix for the N-way classifiers that are trained on responses generated by chat APIs, instruct LLMs, and base LLMs, respectively. The results demonstrate that our classifiers can accurately predict the origin of LLM-generated responses, with minimal confusion between different LLMs.

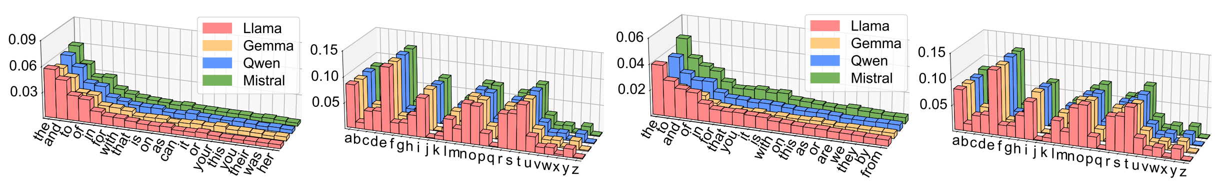


(a) chat APIs (b) instruct LLMs (c) base LLMs

Figure 5.12: Confusion matrices for N-way classifiers on three groups of LLMs: chat APIs, instruct LLMs, and base LLMs.

5.7.6 Words and Letters

Figure 5.13 presents the frequencies of the 20 most commonly used words (*left*) and all English letters (*right*) across instruct and base LLMs. Consistent with our observations in Section 5.4.1, we find notable differences in the distribution of commonly used words between these models, such as "the", "and", "to". In contrast, the letter distributions are nearly identical.



(a) instruct LLMs (b) base LLMs

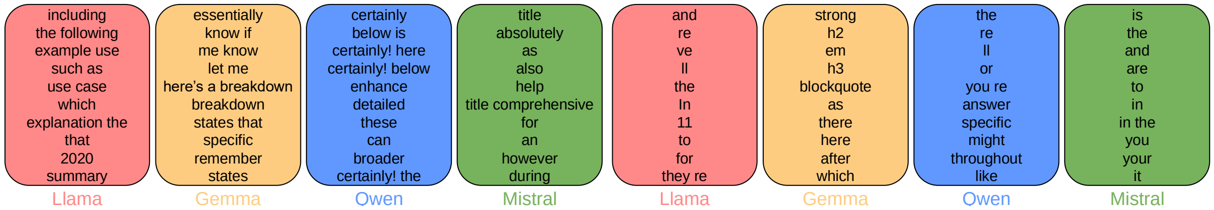
Figure 5.13: Word and letter frequencies in instruct and base LLMs.

5.7.7 Characteristic Phrases

We provide additional results for characteristic phrases as presented in Section 5.4.1. We follow the same methodology in Figure 5.6 to extract characteristic phrases of instruct and base LLMs. Specifically, we train a four-way logistic regression classifier on the TF-IDF features of their responses and use the coefficients to select important phrases of each model.

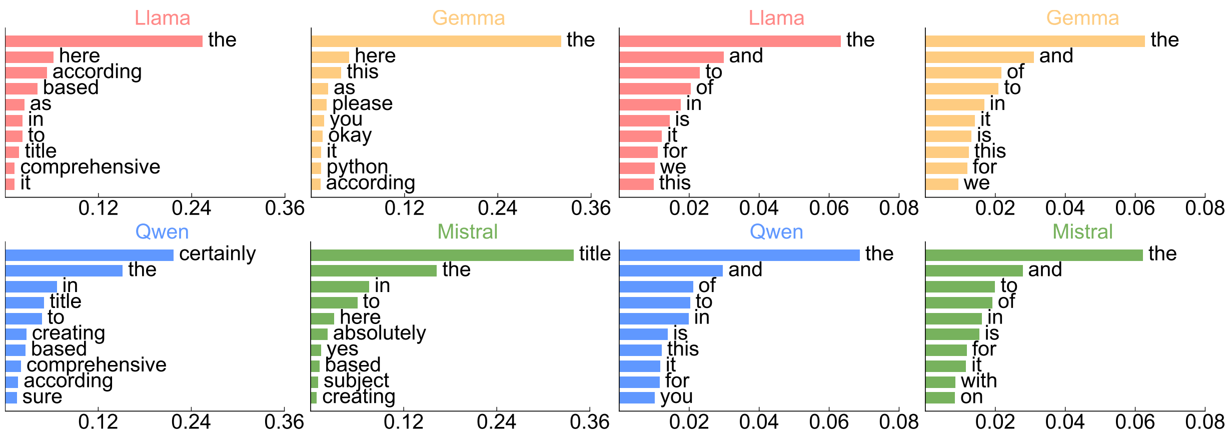
As shown in Figure 5.14, each instruct LLM contains quite distinct characteristic phrases. For example, Llama frequently employs terms "including" and "such as" to introduce specific examples in the output, whereas Gemma tends to engage with users using phrases "let me" and "know if". In contrast, the extracted phrases from base LLMs are less distinctive, primarily consisting of common words such as "the", "to", and "you".

Figure 5.15 illustrates the distribution of first word choices in instruct and base LLMs. Similar to chat APIs (Figure 5.7), instruct LLMs display varied distributions. However, base LLMs exhibit substantial overlap in their most frequent first words, *e.g.*, "the", "and", "of", "to", and "in".



(a) instruct LLMs (b) base LLMs

Figure 5.14: Characteristic phrases for instruct[1] and base LLMs.



(a) instruct LLMs (b) base LLMs

Figure 5.15: Distribution of first word choices in instruct and base LLMs.

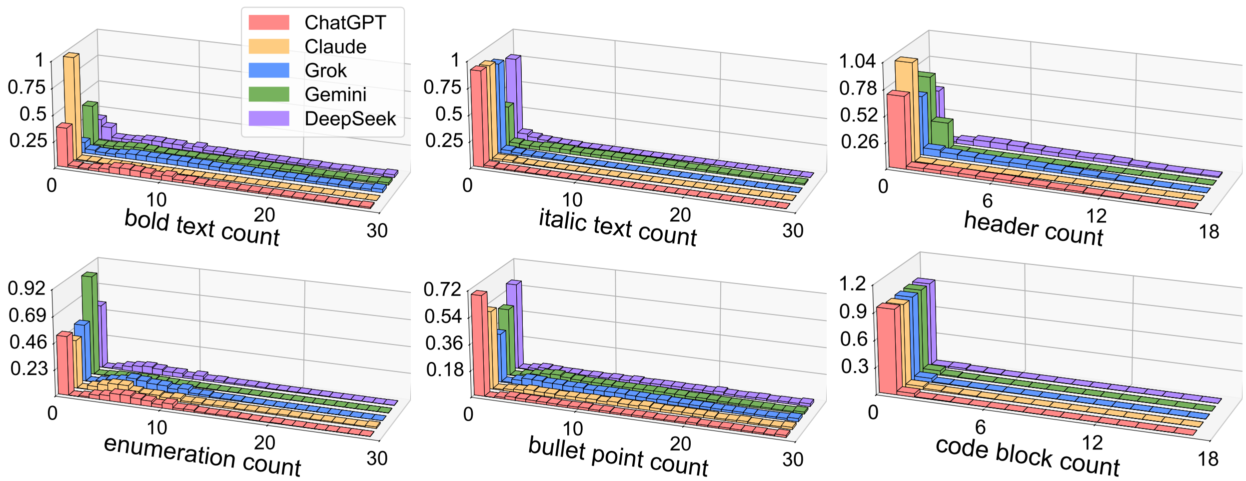
5.7.8 Unique Markdown Formatting

In this part, we provide additional results for the analysis of markdown formatting as presented in Section 5.4.2. Figure 5.16 illustrates the distribution counts of six markdown formatting elements across different models. For both chat API models (Figure 5.16a) and instruct LLMs (Figure 5.16b), we observe distinct differences in the usage of bold texts, headers, enumerations, and bullet points, while italic texts show less variation. Intriguingly, Gemini uses much more italic texts (a lower density at zero in the italic text) than other chat APIs, where similar observations can be found on Gemma2.

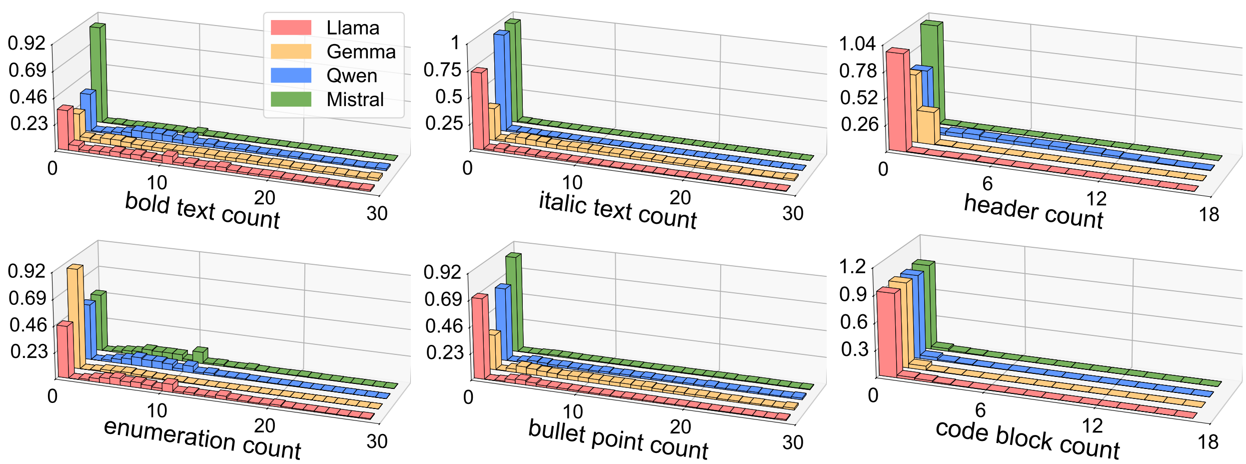
5.7.9 Open-ended Language Analysis

Ablation on LLM judges. Here we demonstrate our findings in Figure 5.9 of Section 5.4.3 remains consistent under several LLM judges. Specifically, we change the LLM judge from ChatGPT to Claude, Grok, and Gemini. We show the results in Figure 5.17. Regardless of the choice of LLM judges, our language analysis reveals that ChatGPT often uses detailed

[1] In LLama of instruct LLMs, the phrase "explanation the" corresponds to a markdown header or bold text for "explanation" followed by a new sentence starting with "the".



(a) chat APIs



(b) instruct LLMs

Figure 5.16: Markdown formatting elements for chat APIs (*top*) and instruct LLMs (*bottom*).

explanations and complex formatting structures, whereas Claude emphasizes key contents without extensive elaboration.

Open-ended language analysis results on other LLMs. In Section 5.4.3, we presented the results of open-ended language analysis for ChatGPT and Claude. Here, we extend our analysis to other chat API models and instruct LLMs. The full results are shown in Figure 5.18, where we use ChatGPT as the LLM judge to compare responses generated by two models within the same category (chat APIs / instruct LLMs). Our analysis highlights several interesting characteristics of each model. For example, Grok's responses tend to feature rich language and comprehensive content, whereas Gemini's outputs are more concise with direct openings.

• Tone: Authoritative and formal, using academic language with detailed explanations and complex sentence structures
• Word Choice: Rich descriptive vocabulary with technical terms, metaphors, and evaluative language ("renowned," "instrumental," "exquisite")
• Opening Style: Direct statements or enthusiastic "Certainly!" responses that establish context and preview content
• Markdown Usage: Extensive formatting with bold headers, nested bullet points, and hierarchical section markers
• Content Structure: Comprehensive paragraphs with detailed explanations, examples, and flowing transitions between ideas

ChatGPT

• Tone: Analytical and cautious, frequently referencing source material and acknowledging limitations
• Word Choice: Simpler, more practical vocabulary focused on clarity and accessibility rather than style
• Opening Style: Meta-references like "According to the text" or explicit disclaimers about content limitations
• Markdown Usage: Minimal formatting with basic bullet points or simple numbered lists
• Content Structure: Concise bullet points and brief statements organized in categorical lists without extensive elaboration

Claude

(a) Claude as the LLM judge.

- **Tone: Warm and Informative** - Uses an inviting, educational tone to engage and inform the reader.
- **Word Choices: Descriptive and Detailed** - Employs rich, technical, or emotive language to enhance understanding or evoke emotion.
- **Opening Lines: Contextual and Engaging** - Provides context or a narrative hook to draw readers in.
- **Markdown Formatting: Structured and Emphasized** - Extensive use of headings, lists, and bold text for clarity and organization.
- **Content: Comprehensive and Insightful** - Focuses on in-depth explanations, examples, and broad implications.

ChatGPT

- **Tone: Direct and Practical** - Often concise, focusing on actionable insights or straightforward information.
- **Word Choices: Simple and Functional** - Prefers straightforward language, emphasizing utility over emotional or detailed descriptions.
- **Opening Lines: Immediate and Focused** - Jumps directly into the topic or lists key points without much preamble.
- **Markdown Formatting: Minimalist** - Uses basic markdown elements like bullet points or simple lists for clarity.
- **Content: Succinct and Focused** - Provides essential information with less depth, often aiming for quick readability or practicality.

Claude

(b) Grok as the LLM judge.

• **Formal Tone:** Often adopts a formal, academic, or instructional tone, prioritizing clarity and detail.
• **Precise Diction:** Employs specific, descriptive, and sometimes technical vocabulary relevant to the subject matter.
• **Contextual Openings:** Typically begins with introductory statements, context-setting, or clear purpose statements.
• **Structured Formatting:** Frequently utilizes headings, lists, bolding, and other markdown features for organization and emphasis.
• **Comprehensive Content:** Provides detailed explanations, examples, and in-depth information, often covering a broader scope.

ChatGPT

• **Informal Tone:** Generally uses a more informal, conversational, or concise tone, prioritizing brevity and accessibility.
• **Simplified Language:** Favors simpler, more direct vocabulary, avoiding jargon and focusing on key information.
• **Direct Openings:** Often starts directly with the main point or a summary of key information.
• **Minimal Formatting:** Uses less markdown formatting, relying primarily on basic lists, bullet points, or plain text.
• **Concise Content:** Focuses on summarizing key points, offering practical advice, or providing concise overviews.

Claude

(c) Gemini as the LLM judge.

Figure 5.17: Results of our open-ended language analysis on ChatGPT and **Claude** with different LLM judges.

5.7.10 Response Demonstrations

In this part, we present examples of LLM responses. Table 5.12 and 5.13 illustrate responses before and after our prompt-level interventions (Section 5.3.2). Table 5.14 shows the transformed responses when only markdown elements are retained (Section 5.4.2). Table 5.15 presents the rewritten responses (Section 5.4.3). Additionally, Table 5.16 provides a pairwise comparison of responses, supporting our findings in Table 5.9. Table 5.17, 5.18, 5.19, 5.20, 5.21, 5.22, 5.23, 5.24, 5.25, 5.26, 5.27, 5.28, 5.29 contain example responses from each LLM considered in Section 5.3, including ChatGPT, Claude, Grok, Gemini, DeepSeek, LLama3.1-8b (instruct), Gemma2-9b (instruct), Qwen2.5-7b (instruct), Mistral-v3-7b (instruct), LLama3.1-8b (base), Gemma2-9b (base), Qwen2.5-7b (base), and Mistral-v3-7b (base).

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices.
If you're looking for cheese alternatives to Brie for your recipe, consider the following options: 1. **Camembert**: This cheese is quite similar to Brie in texture and flavor, offering a creamy consistency and a mild, earthy taste that will complement the beef and apple. 2. **Taleggio**: An Italian cheese with a soft texture and a tangy, fruity flavor. It melts well and can add a unique twist to your dish. 3. **Fontina**: Known for its smooth melting properties, Fontina has a nutty and slightly sweet flavor that pairs nicely with beef and apples. 4. **Goat Cheese**: If you prefer a tangier option, goat cheese can provide a creamy texture and a distinct flavor that contrasts well with the sweetness of the apple. 5. **Havarti**: A semi-soft cheese with a buttery and slightly sweet taste, Havarti melts beautifully and can be a great substitute for Brie. These alternatives should work well in your recipe, providing a delicious and melty addition to your beef and apple combination.

(a) original

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices. Please provide a concise response in a single paragraph, limited to a maximum of 100 words.
For a delicious alternative to Brie, consider using Camembert or triple cream cheese, both of which offer a similar creamy texture and rich flavor. If you prefer a slightly tangier option, goat cheese can provide a delightful contrast to the beef and apple. For a milder taste, Havarti or Fontina can melt beautifully and complement the dish's flavors. Each of these cheeses will add a unique twist while maintaining the creamy, melt-in-your-mouth experience that Brie provides.

(b) length control

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices. Please provide your response in plain text only, avoiding the use of italicized or bold text, lists, markdown, or HTML formatting.
You can use Camembert or triple cream cheese as alternatives to Brie in your recipe. Both have a similar creamy texture and rich flavor that will complement the beef and other ingredients.

(c) format control

Table 5.12: Examples of prompt-level interventions on ChatGPT.

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices.
If you're looking for alternatives to Brie cheese in your recipe, here are some options you can consider: 1. **Goat Cheese**: A soft and creamy cheese with a tangy flavor that pairs well with the sweetness of the apple and the savory flavor of the beef. 2. **Feta Cheese**: A salty and crumbly cheese that adds a nice contrast to the softness of the beef and the crunch of the apple. 3. **Camembert**: A French cheese with a creamy texture and a mild, earthy flavor that complements the beef and apple nicely. 4. **Blue Cheese**: A strong and pungent cheese that adds a bold flavor to the dish. Use a milder blue cheese if you're not a fan of strong flavors. 5. **Mascarpone**: A rich and creamy cheese that adds a luxurious touch to the dish. It's a good option if you want a milder flavor than Brie. 6. **Ricotta**: A soft and creamy cheese that's a good option if you want a lighter flavor than Brie. It's also a good choice if you're looking for a lower-calorie alternative. 7. **Taleggio**: An Italian cheese with a creamy texture and a mild, fruity flavor that pairs well with the apple and beef. When substituting Brie, keep the following in mind: • If using a stronger cheese like Blue Cheese or Goat Cheese, use a smaller amount to avoid overpowering the other flavors. • If using a milder cheese like Mascarpone or Ricotta, you may want to add a bit more to achieve the desired creaminess. • If using a crumbly cheese like Feta, you may want to crumble it instead of slicing it to distribute the flavor evenly. Remember, the key is to balance the flavors and textures in your dish, so feel free to experiment and find the combination that works best for you!

(a) original

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices. Please provide a concise response in a single paragraph, limited to a maximum of 100 words.
If you're looking for alternatives to Brie, consider using other soft-ripened cheeses like Camembert, Goat Cheese, or Feta. Alternatively, you could opt for a semi-soft cheese like Havarti or Muenster. For a stronger flavor, try using Blue Cheese or Gorgonzola. If you prefer a milder taste, Boursin or Mascarpone could work well. Each of these cheeses will melt and complement the beef and apple nicely, adding a unique twist to your dish. Simply replace the Brie with your chosen cheese and proceed with the recipe as instructed.

(b) length control

Can you suggest any other cheese alternatives that I can use instead of Brie for this recipe? Generate according to: Add one or two slices Brie o the beef and let it melt. Cut the green apple in thin slices. Top with the mixed salad and finish with the apple slices. Please provide your response in plain text only, avoiding the use of italicized or bold text, lists, markdown, or HTML formatting.
You can consider the following cheese alternatives to Brie in your recipe: Camembert: This French cheese has a similar creamy texture and mild, earthy flavor to Brie, making it a suitable substitute. Goat cheese: A tangy and creamy cheese that pairs well with the sweetness of the apple and the savory flavor of the beef. Feta: A crumbly Greek cheese that adds a salty flavor to the dish, which can balance out the sweetness of the apple. Blue cheese: A strong and pungent cheese that can add a bold flavor to the dish, but be aware that it may overpower the other ingredients. Ricotta: A mild and creamy cheese that can add a light and airy texture to the dish, but it may not provide the same richness as Brie. Mascarpone: An Italian cheese with a rich and creamy texture that can add a luxurious feel to the dish, but it may be sweeter than Brie. Keep in mind that each of these alternatives will change the flavor profile of the dish, so you may need to adjust the other ingredients accordingly.

(c) format control

Table 5.13: Examples of prompt-level interventions on Llama3.1-8b-Instruct.

What are some effective tactics for building a referral network and gaining new business through word-of-mouth?

Building a strong referral network and gaining new business through word-of-mouth can be highly effective and cost-efficient. Here are some tactics to achieve this:

1. **Deliver Exceptional Service**: The foundation of word-of-mouth marketing is consistent excellence. Providing top-notch services or products encourages satisfied customers to share their positive experiences with others.
2. **Identify and Nurture Key Relationships**: Focus on building strong relationships with clients, partners, and industry peers who can refer you to others. Personalize your interactions and maintain regular communication to keep these relationships strong.
3. **Ask for Referrals**: Don't hesitate to ask satisfied clients for referrals. Timing is crucial—make the request at the peak of their satisfaction. Be clear about the type of referrals you are seeking.
4. **Referral Incentives**: Offer incentives such as discounts, gift cards, or exclusive access to services to encourage customers or partners to refer new business to you. Ensure these incentives are enticing enough to motivate action.
5. **Leverage Social Proof**: Encourage satisfied customers to leave reviews or testimonials online. Positive reviews on platforms such as Google, Yelp, or LinkedIn can influence potential clients.
6. **Engage on Social Media**: Use social media platforms to share success stories, client testimonials, and updates about your business. Engage with your audience through comments, shares, and likes to build a community around your brand.
7. **Host Events**: Organize events such as workshops, webinars, or networking meet-ups. These events provide value, showcase your expertise, and encourage attendees to refer you to others.
8. **Partner with Complementary Businesses**: Collaborate with businesses that offer complementary services or products. This creates mutual referral opportunities and extends your reach to their customer base.
9. **Create a Referral Program**: Formalize your referral process with a structured program, ensuring referrers are aware of the benefits and the procedure. Clearly communicate the value proposition and streamline the referral process.
10. **Stay Top-of-Mind**: Regularly connect with your network through emails, newsletters, or personalized messages. Share updates, industry insights, or tips that position you as a valuable and knowledgeable resource.
11. **Thank Referrers**: Always express gratitude to those who refer new business to you. Acknowledging their support strengthens relationships and encourages further referrals.
12. **Measure and Optimize**: Track where your referrals are coming from and the effectiveness of different tactics. Use this data to optimize your approach, focusing on strategies that yield the best results.

By implementing these tactics and consistently nurturing your network, you can effectively build a robust referral system that generates new business through word-of-mouth recommendations.

(a) original

xxx

1. **xxx**xxx
2. **xxx**xxx
3. **xxx**xxx
4. **xxx**xxx
5. **xxx**xxx
6. **xxx**xxx
7. **xxx**xxx
8. **xxx**xxx
9. **xxx**xxx
10. **xxx**xxx
11. **xxx**xxx
12. **xxx**xxx

xxx

(b) markdown elements only

Table 5.14: Examples of only using markdown formatting elements and replacing text content with "xxx" placeholders.

- **Descriptive Tone:** Emphasizes detail, narrative, or objective perspectives, often exploring themes with depth and nuance.
- **Rich Language:** Incorporates emotive, analytical, or descriptive language to enhance understanding and engagement.
- **Detailed Openings:** Introduces the subject with context, thematic elements, or guiding points, establishing a comprehensive view.
- **Structured Formatting:** Utilizes formatting elements like headings, lists, and bold text to organize information clearly.
- **Comprehensive Content:** Delivers in-depth analysis or exploration of subjects, highlighting diverse factors or broader implications.

Grok

- **Concise Tone:** Often adopts a straightforward, direct, or expressive approach focusing on succinctness and dramatic tension.
- **Specific Language:** Utilizes precise or vivid language to convey themes like tension, urgency, and significant events or outcomes.
- **Direct Openings:** Typically begins with immediate context or action, setting a focused tone or premise from the start.
- **Simple Formatting:** Relies mainly on plain text or minimal formatting to maintain clarity and directness.
- **Core Content:** Centers on key insights or motivations, providing essential information or narratives with limited embellishment.

Gemini

(a) chat APIs

- **Tone Characterization:** Often formal, factual, and exacting, emphasizing detailed analysis and structured presentations.
- **Word Choices Depiction:** Technical and precise, leveraging industry-specific or academic terminology to convey depth and credibility.
- **Opening Line Initiation:** Frequently starts directly with context, setting an immediate formal or informative tone.
- **Markdown Use:** Prefers structured formats with headings, bullet points, or numbered lists to enhance clarity.
- **Content Focus:** Delivers comprehensive, detailed content with substantial data, examples, or specific tactics.

Llama

- **Tone Characterization:** Generally more conversational, engaging, and accessible, often blending storytelling or personal insights.
- **Word Choices Depiction:** Simple, expressive, and often emotive language, creating relatability and narrative flair.
- **Opening Line Initiation:** Begins engagingly, often with context or an emotional hook, to capture the reader's interest.
- **Markdown Use:** Employs varied markdown elements like bullet points or emphasis for readability and structure.
- **Content Focus:** Centers on overarching concepts, insights, and experiences, often prioritizing engagement over exhaustive detail.

Gemma

- **Detailed and Analytical Tone:** The tone is often formal, focusing on analysis, exploration, and organized narratives.
- **Descriptive and Formal Word Choices:** Uses technical and descriptive vocabulary that highlights complexity and depth.
- **Comprehensive Opening Lines:** Starts often with broad context or detailed introductions, setting a formal framework.
- **Structured Markdown Formatting:** Utilizes headers, sections, and lists to enhance readability and organization.
- **Content Depth and Exploration:** Provides comprehensive narratives with detailed discussions, focusing on systematic approaches and analysis.

Qwen

- **Concise and Direct Tone:** The tone is generally straightforward, practical, and sometimes conversational, focusing on direct explanations.
- **Simplified and Emotive Word Choices:** Prefers straightforward language, occasionally prioritizing emotional or motivational impact.
- **Immediate Opening Lines:** Begins directly with subject matter or actions, often in conversational or narrative styles.
- **Minimalist Markdown Formatting:** Uses simpler formatting with lists or plain text to maintain clarity without embellishments.
- **Summary Focused Content:** Concentrates on key points with succinct explanations, prioritizing clarity and accessibility over comprehensive detail.

Mistral

(b) instruct LLMs

Figure 5.18: Additional results of our open-ended language analysis on chat APIs (*top*) and instruct LLMs (*bottom*).

What are some ways to cope with feelings of loneliness or isolation?

Coping with feelings of loneliness or isolation can be challenging, but there are several strategies that might help:

1. **Connect with Others**: Reach out to friends and family through phone calls, video chats, or messaging apps. Joining clubs, groups, or online communities related to your interests can also help you meet new people.
2. **Volunteer**: Helping others can provide a sense of purpose and community. Look for local organizations or virtual opportunities where you can volunteer your time or skills.
3. **Engage in Hobbies**: Pursue activities or hobbies that you enjoy, whether it's reading, painting, cooking, or playing a musical instrument. These can be fulfilling and help distract from loneliness.
4. **Physical Activity**: Exercise is known to boost mood and alleviate stress. Try walking, jogging, yoga, or any form of physical activity that you enjoy.
5. **Mindfulness and Meditation**: Practices such as meditation, deep breathing, or mindfulness can help anchor you in the present moment and reduce feelings of loneliness.
6. **Professional Help**: Consider speaking with a therapist or counselor, who can provide support and coping strategies tailored to your situation.
7. **Pet Companionship**: If feasible, consider getting a pet. Animals can offer companionship and can be a source of comfort and joy.
8. **Limit Social Media**: While it might seem connecting, excessive use of social media can sometimes exacerbate feelings of loneliness. Set boundaries for healthier use.
9. **Self-Compassion**: Practice being kind to yourself. Acknowledge your feelings without judgment and remind yourself that it's okay to feel lonely at times.
10. **Set Goals**: Setting small, achievable goals can provide a sense of accomplishment and forward momentum, fostering a sense of purpose.
11. **Seeking Community Spaces**: Libraries, community centers, or cafes can be great places to be around others, even if you're not directly interacting with them.

Remember, everyone experiences loneliness differently, and it's important to find what works best for you.

(a) original

Dealing with feelings of loneliness or isolation can be tough, but there are various strategies that may help:

1. **Reach Out**: Contact friends and family through phone calls, video chats, or messaging platforms. Joining clubs, groups, or online communities that align with your interests can also help you connect with new people.
2. **Volunteer Your Time**: Helping others can give you a sense of purpose and community. Look for local charities or online opportunities to contribute your time or skills.
3. **Pursue Your Hobbies**: Engage in activities you enjoy, such as reading, painting, cooking, or playing a musical instrument. These pursuits can be satisfying and provide a distraction from feelings of loneliness.
4. **Stay Active**: Regular exercise is known to improve mood and reduce stress. Try activities like walking, jogging, yoga, or any form of physical exercise that appeals to you.
5. **Practice Mindfulness**: Techniques such as meditation, deep breathing, or mindfulness can help ground you in the present and alleviate feelings of loneliness.
6. **Consider Professional Guidance**: Talking to a therapist or counselor can provide helpful support and coping techniques tailored to your needs.
7. **Get a Pet**: If it's an option for you, think about adopting a pet. Animals can be wonderful companions and bring joy and comfort into your life.
8. **Limit Social Media Use**: Though it may seem like a way to connect, extensive use of social media can sometimes increase feelings of loneliness. Establish limits for a healthier engagement.
9. **Be Kind to Yourself**: Practice self-compassion by acknowledging your feelings without judgment and reminding yourself that it's normal to feel lonely occasionally.
10. **Set Achievable Goals**: Creating small, realistic goals can foster a sense of accomplishment and help you move forward, enhancing your sense of purpose.
11. **Use Community Spaces**: Places like libraries, community centers, or cafes can provide an opportunity to be around others, even if you're not directly interacting with them.

Remember, everyone experiences loneliness in different ways, so it's essential to discover what strategies work best for you.

(b) paraphrasing

Coping with loneliness can be challenging, but several strategies can help. Connecting with friends and family, volunteering, and engaging in enjoyable hobbies can foster a sense of community. Physical activity and mindfulness practices can improve mood and reduce stress. Seeking professional help and considering companionship from pets may also provide comfort. It's essential to limit social media usage, practice self-compassion, set small goals for achievement, and explore community spaces like libraries or cafes. Ultimately, finding individualized approaches that resonate with you is key to managing feelings of loneliness.

(c) summarizing

Table 5.15: Examples of LLM rewriting using GPT-4o-mini.

What are some ways to cope with feelings of loneliness or isolation?

Coping with feelings of loneliness or isolation can be challenging, but there are several strategies that might help:

1. **Connect with Others**: Reach out to friends and family through phone calls, video chats, or messaging apps. Joining clubs, groups, or online communities related to your interests can also help you meet new people.
2. **Volunteer**: Helping others can provide a sense of purpose and community. Look for local organizations or virtual opportunities where you can volunteer your time or skills.
3. **Engage in Hobbies**: Pursue activities or hobbies that you enjoy, whether it's reading, painting, cooking, or playing a musical instrument. These can be fulfilling and help distract from loneliness.
4. **Physical Activity**: Exercise is known to boost mood and alleviate stress. Try walking, jogging, yoga, or any form of physical activity that you enjoy.
5. **Mindfulness and Meditation**: Practices such as meditation, deep breathing, or mindfulness can help anchor you in the present moment and reduce feelings of loneliness.
6. **Professional Help**: Consider speaking with a therapist or counselor, who can provide support and coping strategies tailored to your situation.
7. **Pet Companionship**: If feasible, consider getting a pet. Animals can offer companionship and can be a source of comfort and joy.
8. **Limit Social Media**: While it might seem connecting, excessive use of social media can sometimes exacerbate feelings of loneliness. Set boundaries for healthier use.
9. **Self-Compassion**: Practice being kind to yourself. Acknowledge your feelings without judgment and remind yourself that it's okay to feel lonely at times.
10. **Set Goals**: Setting small, achievable goals can provide a sense of accomplishment and forward momentum, fostering a sense of purpose.
11. **Seeking Community Spaces**: Libraries, community centers, or cafes can be great places to be around others, even if you're not directly interacting with them.

Remember, everyone experiences loneliness differently, and it's important to find what works best for you.

(a) ChatGPT

There are several healthy ways to cope with loneliness and isolation:

1. Maintain social connections:

- Call or video chat with friends and family
- Send messages, emails, or letters
- Join online communities or social media groups
- Schedule regular virtual meetups

2. Pursue hobbies and interests:

- Take up a new hobby
- Join clubs or groups related to your interests
- Take online classes
- Learn a new skill

3. Physical activities:

- Exercise regularly
- Take walks outdoors
- Practice yoga or meditation
- Stay physically active

4. Connect with your community:

- Volunteer
- Join local organizations
- Attend community events
- Participate in religious or spiritual groups

5. Self-care practices:

- Maintain a routine
- Practice mindfulness
- Keep a journal
- Get enough sleep
- Eat healthy meals

6. Seek professional help:

- Talk to a therapist or counselor
- Join support groups
- Consider online therapy options

7. Adopt or care for a pet:

- Pets provide companionship
- Create a sense of purpose
- Encourage routine and responsibility

Remember that feeling lonely is a common human experience, and it's okay to reach out for help when needed.

(b) Claude

Table 5.16: Examples from ChatGPT and Claude that illustrate results of our open-ended language analysis in Figure 5.9.

Can polar bears migrate to new habitats if their Arctic environment changes too drastically?
Polar bears are highly specialized animals adapted to the Arctic environment, primarily relying on sea ice for hunting seals, their main prey. If their Arctic environment changes drastically, such as through significant sea ice loss due to climate change, their ability to migrate to new habitats is limited by several factors: 1. **Species Adaptation**: Polar bears are uniquely adapted to life on sea ice and in cold climates. They have thick fur, a layer of fat for insulation, and physical characteristics like large paws for swimming and walking on ice. These adaptations make it difficult for them to survive in non-polar environments. 2. **Dietary Needs**: Their diet consists mainly of seals, which they hunt on sea ice. New habitats may not support a sufficient seal population or alternative prey that polar bears can easily exploit. 3. **Geographical Barriers**: Polar bears are largely confined to the Arctic and areas surrounding the North Pole. There are no direct routes to migrate to other cold environments that would support them in a similar way. 4. **Competition and Predation**: Moving to new habitats would likely involve entering territories with existing ecosystems that include competition for food from other predators or unfamiliar threats. 5. **Habitat Fragmentation**: The fragmentation of ice habitats does not create viable paths for migration but rather isolates populations, reducing genetic diversity and making survival more precarious. In conclusion, while polar bears might be able to make small relocations within the Arctic as sea ice conditions shift, extensive migration to entirely new habitats is not feasible due to their specific adaptations and the absence of suitably similar environments elsewhere. Conservation efforts focus more on mitigating climate change and preserving their current habitat.

Table 5.17: Example response of ChatGPT.

Can polar bears migrate to new habitats if their Arctic environment changes too drastically?
Polar bears have limited ability to migrate to new habitats if Arctic conditions become unsuitable. They are highly specialized for life on sea ice and depend on it for hunting seals, their primary food source. While they can swim long distances, moving to southern territories would be challenging because: 1) they are not well-adapted to warmer climates, 2) their hunting techniques are specific to ice-based prey, and 3) competition with other bear species and humans would increase. Their survival is closely tied to maintaining Arctic sea ice habitat.

Table 5.18: Example response of Claude.

Can polar bears migrate to new habitats if their Arctic environment changes too drastically?
Polar bears are highly specialized for their Arctic environment, and their ability to migrate to entirely new habitats is limited by several factors: 1. **Adaptation**: Polar bears are adapted to cold environments with a diet primarily consisting of seals, which they hunt from sea ice. Their thick fur, large size, and fat reserves are suited for Arctic conditions, not temperate or tropical climates. 2. **Habitat Requirements**: They need sea ice for hunting, breeding, and resting. Without ice, their ability to catch seals, which is their primary food source, is severely compromised. Alternative habitats would need to provide similar conditions or abundant alternative prey, which is rare outside the Arctic. 3. **Physiological Limitations**: Polar bears might find it challenging to adapt to significantly different temperatures or ecosystems. For instance, their large bodies and fur make them prone to overheating in warmer climates. 4. **Behavioral Flexibility**: While bears in general can be quite adaptable in terms of diet (e.g., grizzly bears), polar bears have evolved in an environment with very specific conditions. Their behavioral adaptations are finely tuned to the Arctic, reducing the likelihood of successful migration to drastically different environments. 5. **Human Activity**: As polar bears move southward in search of food or better conditions, they increasingly encounter human populations and infrastructure, leading to conflicts, habitat fragmentation, and often, negative outcomes for the bears. 6. **Genetic Considerations**: Polar bears have evolved in relative isolation in the Arctic. Mixing with other bear species or adapting to entirely new environments could lead to genetic issues or dilution of their specialized adaptations. However, there are some considerations: • **Hybridization**: There's evidence of polar bears mating with grizzly bears (creating "pizzly" or "grolar" bears) in areas where their ranges overlap due to climate change. This might be seen as a form of adaptation or migration at a genetic level, though it's not migration in the traditional sense. • **Range Expansion**: Some polar bears have been observed traveling further south than usual, likely in search of food as ice recedes. However, this is more about extending their current habitat rather than migrating to a completely different one. • **Conservation Efforts**: Efforts might focus on preserving or creating corridors where polar bears can move between suitable habitats or on habitat restoration to ensure the Arctic remains viable for them. In summary, while polar bears could potentially expand their range or adapt behaviorally to some extent, true migration to entirely new, non-Arctic habitats is unlikely due to their specific ecological needs and physiological adaptations. Conservation efforts are more focused on mitigating the impacts of climate change in their current habitat or facilitating their movement within the Arctic region.

Table 5.19: Example response of Grok.

Can polar bears migrate to new habitats if their Arctic environment changes too drastically?
While polar bears have some capacity for movement and adaptation, large-scale migration to entirely new habitats in response to Arctic change is highly unlikely and faces significant obstacles: • **Physiological Specialization:** Polar bears are highly specialized for Arctic life. Their thick blubber, fur, and hunting techniques are specifically adapted for cold, icy environments. They rely heavily on sea ice for hunting seals, their primary food source. Relocating to warmer environments with different prey and terrain would be extremely challenging. • **Competition and Existing Ecosystems:** Other bear species or large predators already occupy potential alternative habitats. Introducing polar bears would create competition for resources and potentially disrupt existing ecosystems. Polar bears are apex predators, and their presence would have cascading effects. • **Limited Land-Based Food Sources:** While polar bears can sometimes scavenge on land, terrestrial food sources are insufficient to sustain their large body size and energy demands. They are primarily dependent on the rich calorie content of seals. Land-based food would not be abundant enough to support large populations. • **Human-Wildlife Conflict:** As polar bears are forced further south or into more populated areas in search of food due to shrinking sea ice, encounters with humans are likely to increase, leading to conflict and potential danger for both bears and people. Relocation efforts could exacerbate this issue. • **Distance and Barriers:** Potential alternative habitats are often geographically distant from the Arctic. Natural barriers such as mountains, oceans, and unsuitable terrain would make migration extremely difficult. Furthermore, suitable habitat may already be occupied by other species or fragmented by human development. • **Genetic Adaptation Takes Time:** While some level of adaptation is possible within a species, evolving to thrive in a completely different environment takes many generations. The rapid pace of Arctic change is outpacing the ability of polar bears to adapt genetically to significantly different conditions. Instead of large-scale migration, polar bears are more likely to exhibit smaller-scale shifts in range, dietary changes (though with limited success), and increased hybridization with grizzly bears. However, the long-term survival of the species ultimately depends on mitigating climate change and preserving their Arctic habitat.

Table 5.20: Example response of Gemini.

Can polar bears migrate to new habitats if their Arctic environment changes too drastically?
Polar bears are highly specialized for life in the Arctic, relying on sea ice to hunt their primary prey, seals. If the Arctic environment changes too drastically, particularly due to climate change and the loss of sea ice, polar bears would face significant challenges in migrating to new habitats. Here are some key points to consider: 1. **Specialization**: Polar bears are adapted to the cold, icy conditions of the Arctic. Their physical and behavioral traits are finely tuned to this environment, making it difficult for them to thrive in vastly different habitats. 2. **Prey Availability**: Polar bears primarily hunt seals, which are also dependent on sea ice. If sea ice diminishes, both polar bears and their prey would be affected. Moving to new habitats would require finding alternative food sources, which may not be as abundant or accessible. 3. **Geographic Barriers**: The Arctic is relatively isolated, and polar bears would face significant geographic barriers if they attempted to migrate to new areas. Moving southward would bring them into competition with other bear species, such as grizzly bears, and into environments that are not suited to their specific needs. 4. **Human Impact**: Many potential new habitats for polar bears are already occupied by humans. Human activities, such as urban development, agriculture, and industry, could pose additional challenges for polar bears trying to adapt to new areas. 5. **Genetic Adaptation**: Over long periods, species can adapt to new environments through genetic changes. However, the rapid pace of climate change may outstrip the ability of polar bears to adapt quickly enough to survive in significantly altered or new habitats. 6. **Conservation Efforts**: Conservation strategies are being developed to help polar bears cope with changing environments. These include protecting critical habitats, reducing greenhouse gas emissions to mitigate climate change, and researching potential relocation or assisted migration options. In summary, while polar bears have some capacity to adapt to changing conditions, their specialized nature and the rapid pace of environmental change pose significant challenges to their ability to migrate to and thrive in new habitats. Conservation efforts are crucial to ensuring their survival in the face of these challenges.

Table 5.21: Example response of DeepSeek

What are some effective tactics for building a referral network and gaining new business through word-of-mouth?
Building a referral network and leveraging word-of-mouth marketing can be a powerful way to grow your business. Here are some effective tactics to help you achieve this: 1. **Deliver Exceptional Service**: The foundation of a strong referral network is delivering exceptional service to your existing customers. Ensure that you consistently meet or exceed their expectations, and they will be more likely to refer you to others. 2. **Build Relationships**: Foster strong relationships with your customers, partners, and colleagues. This can be done through regular communication, shared experiences, and mutual support. 3. **Encourage Referrals**: Make it easy for your customers to refer you by providing incentives, such as discounts, rewards, or exclusive offers. You can also ask for referrals directly, but be sure to do so in a non-intrusive and respectful manner. 4. **Leverage Social Proof**: Use customer testimonials, case studies, and reviews to demonstrate your expertise and build credibility. This can be done through social media, your website, or other marketing channels. 5. **Host Events and Webinars**: Hosting events, webinars, or workshops can help you build relationships with potential customers and establish yourself as an expert in your field. 6. **Partner with Other Businesses**: Collaborate with complementary businesses to expand your network and reach new audiences. 7. **Use Referral Marketing Platforms**: Utilize platforms like ReferralCandy, Ambassador, or B2B referral platforms to streamline your referral process and track referrals. 8. **Follow Up**: Follow up with your referrals and leads to ensure they have a positive experience and to gather feedback. 9. **Be Authentic and Transparent**: Be genuine and transparent in your interactions with your network. This will help build trust and credibility, making it more likely that people will refer you to others. 10. **Measure and Analyze**: Track the effectiveness of your referral efforts and analyze the data to refine your strategy and optimize your results. Additionally, consider the following: • **Employee Advocacy**: Encourage your employees to become brand ambassadors and share their positive experiences with their networks. • **Customer Advocacy**: Identify and empower your most loyal customers to become advocates for your business. • **Content Marketing**: Create valuable content that educates and entertains your audience, making them more likely to share it with others. • **Influencer Marketing**: Partner with influencers in your industry to expand your reach and credibility. By implementing these tactics, you can build a strong referral network and leverage word-of-mouth marketing to drive growth and success for your business.

Table 5.22: Example response of Llama3.1-8b (instruct).

What are some effective tactics for building a referral network and gaining new business through word-of-mouth?

Building a Referral Network & Gaining New Business Through Word-of-Mouth:
1. Deliver Exceptional Service:

- **Exceed expectations:** Go above and beyond for every client. Surprise and delight them with unexpected value.
- **Build strong relationships:** Get to know your clients personally, understand their needs, and build genuine connections.
- **Be responsive and reliable:** Respond promptly to inquiries, meet deadlines, and follow through on promises.

2. Make it Easy to Refer:

- **Create a referral program:** Offer incentives for successful referrals, such as discounts, free services, or exclusive perks.
- **Provide referral materials:** Make it easy for clients to share your information with others by providing referral cards, social media share buttons, and email templates.
- **Streamline the process:** Make it simple for clients to refer friends and family with a clear and concise process.

3. Nurture Your Network:

- **Stay in touch:** Regularly communicate with your existing clients through email newsletters, social media updates, or personalized messages.
- **Attend industry events:** Network with other professionals and potential referral partners.
- **Join relevant online communities:** Engage in online forums and groups related to your industry.

4. Leverage Social Media:

- **Share valuable content:** Provide helpful tips, insights, and resources that your target audience will find valuable.
- **Encourage user-generated content:** Ask clients to share their experiences with your business on social media.
- **Run contests and giveaways:** Offer incentives for people to share your content and refer their friends.

5. Ask for Referrals:

- **Don't be afraid to ask:** Politely request referrals from satisfied clients.
- **Make it personal:** Explain why you value their referrals and how they can help your business grow.
- **Offer specific suggestions:** Provide examples of people who might benefit from your services.

6. Track Your Results:

- **Monitor your referral rate:** Track the number of referrals you receive and the conversion rate of those referrals.
- **Analyze your data:** Identify which tactics are most effective and adjust your strategy accordingly.
- **Celebrate your successes:** Recognize and reward your top referrers.

Remember, building a strong referral network takes time and effort. Be patient, consistent, and focus on providing exceptional value to your clients.

Table 5.23: Example response of Gemma2-9b (instruct).

What are some effective tactics for building a referral network and gaining new business through word-of-mouth?
Building a referral network and leveraging word-of-mouth to gain new business is a powerful strategy. Here are some effective tactics to consider: 1. **Identify Your Ideal Referral Sources:** • Determine who your ideal customers are and who they might trust to recommend your services or products. This could be current customers, industry influencers, or other businesses in your network. 2. **Create a Referral Program:** • Develop a clear and attractive referral program that incentivizes both the referrer and the referred. Incentives can range from discounts, free services, or even just a simple thank you note. 3. **Educate Your Network:** • Provide your network with the necessary information to make a referral. This could include testimonials, case studies, or a simple one-pager that explains the benefits of your product or service. 4. **Make It Easy to Refer:** • Provide easy-to-use tools for your network to make referrals. This could be a referral form, a link, or a simple email template they can use. 5. **Leverage Social Media:** • Use social media platforms to connect with potential referrers and to share your referral program. Platforms like LinkedIn, Twitter, and Facebook can be particularly effective. 6. **Host Events and Networking Opportunities:** • Organize events or participate in networking opportunities where you can meet potential referrers in person. Personal connections can often lead to more referrals. 7. **Provide Excellent Customer Service:** • Ensure that your current customers have a positive experience with your product or service. Satisfied customers are more likely to refer others. 8. **Follow Up and Show Appreciation:** • Keep in touch with your network and show appreciation for their referrals. A simple thank you note or a small token of appreciation can go a long way. 9. **Use Testimonials and Case Studies:** • Share positive testimonials and case studies from satisfied customers. This can help build trust and encourage others to refer to you. 10. **Be Patient and Persistent:** • Building a referral network takes time. Be patient and persistent in your efforts. Consistency is key. 11. **Monitor and Analyze:** • Keep track of your referral program's performance and analyze the data to see what's working and what isn't. Use this information to refine your approach. 12. **Offer Referral Bonuses:** • Consider offering bonuses for repeat referrals or for referring a certain number of new customers. This can help keep the momentum going. By implementing these tactics, you can build a strong referral network and leverage word-of-mouth to grow your business.

Table 5.24: Example response of Qwen2.5-7b (instruct).

What are some effective tactics for building a referral network and gaining new business through word-of-mouth?
1. Provide exceptional service: The foundation of a strong referral network is built on delivering outstanding service to your current clients. Happy customers are more likely to refer you to others. 2. Ask for referrals: Don't be shy about asking satisfied clients for referrals. Make it easy for them by providing referral cards, email templates, or a simple script they can use to introduce you to their contacts. 3. Offer incentives: Consider offering incentives for referrals, such as discounts, freebies, or other rewards. This can motivate your clients to refer more people to your business. 4. Networking: Attend industry events, trade shows, and networking events to meet potential referral sources. Joining professional organizations and participating in local business groups can also help you expand your network. 5. Leverage social media: Use social media platforms to connect with potential clients and engage with your current network. Share valuable content, participate in discussions, and don't forget to ask for referrals on your social media profiles. 6. Follow up: Follow up with leads promptly and professionally. A quick response can make a positive impression and increase the chances of converting a referral into a client. 7. Build relationships: Focus on building long-term relationships with your clients and referral sources. Regularly touch base with them, show appreciation for their referrals, and keep them updated on your business and industry news. 8. Provide referral training: Teach your clients how to effectively refer you to others. This can include providing them with tips on how to introduce you, what information to share, and how to follow up on referrals. 9. Collaborate with non-competitive businesses: Partner with other non-competitive businesses to cross-refer clients. This can help you tap into their network and vice versa. 10. Ask for testimonials: Positive testimonials can help build trust with potential clients and encourage referrals. Ask satisfied clients if they would be willing to provide a testimonial for your website or social media profiles.

Table 5.25: Example response of Mistral-v3-7b (instruct).

Newswise — Sepsis is a serious medical condition caused by an overwhelming immune response to infection. The body's infection-fighting chemicals trigger widespread inflammation
, which can lead to blood clots and leaky blood vessels. It damages the tiny capillaries in organs such as lungs, liver, kidney and brain. According to recent estimates, 1 million people die from sepsis each year. In the United States alone, it affects more than one million hospital patients annually and accounts for over $20 billion in healthcare costs, making it the most expensive condition treated in intensive care units (ICUs). A research team led by Dr. Robert W. Day at Washington University School of Medicine found that a common drug used for treating type 2 diabetes may reduce mortality rates among ICU patients with severe sepsis or septic shock. The researchers examined data on more than 340,000 adult patients admitted to ICUs across the U.S. between January 2015 and December 2017. Approximately 50% were diagnosed with severe sepsis or septic shock. They discovered that the use of metformin – the first-line treatment for type 2 diabetes – was associated with a significant reduction in the risk of death within 30 days after admission to the ICU. The benefit was even greater among those who didn't have pre-existing diabetes. The findings suggest that metformin could be considered as part of standard practice for all ICU patients with severe sepsis or septic shock. Metformin has been shown to improve outcomes in animal models of sepsis. According to the researchers, this study provides further evidence linking metformin use with better survival rates in human subjects experiencing extreme illness.

Table 5.26: Example response of Llama3.1-8b (base).

Newswise — Sepsis is a serious medical condition caused by an overwhelming immune response to infection. The body's infection-fighting chemicals trigger widespread inflammation
, which can lead to organ damage and failure. While sepsis affects only 1% of the U.S. population each year, it kills more than half a million people globally every year—making it as dangerous as cancer or malaria. In fact, sepsis accounts for about one in six deaths worldwide each year. It also ranks among the top five leading causes of death from infectious diseases. When we think of 'sepsis,' we often imagine elderly patients with multiple comorbidities," said Michael Miller, M.D., chief of the Division of Pulmonary, Critical Care & Sleep Medicine at UT Southwestern Medical Center and holder of the Dallas Heart Ball Chair in Cardiovascular Disease Research. "But our research has shown that 40% of people who develop severe sepsis are young adults.

Table 5.27: Example response of Gemma2-9b (base).

Newswise — Sepsis is a serious medical condition caused by an overwhelming immune response to infection. The body's infection-fighting chemicals trigger widespread inflammation,
which can damage healthy organs and blood vessels. Sepsis is treated in the hospital with antibiotics and fluids. If sepsis isn't detected early or if treatment doesn't work, multiple organ failure occurs and may lead to death.

Table 5.28: Example response of Qwen2.5-7b (base).

Newswise — Sepsis is a serious medical condition caused by an overwhelming immune response to infection. The body's infection-fighting chemicals
, which are normally good, can cause damage to the blood vessels and organs. Sepsis is one of the leading causes of death in hospitals. Although there have been recent advances in sepsis treatment, researchers are still working hard to identify new ways to prevent and treat this complex disease. Researchers at Washington University School of Medicine in St. Louis recently published a study that may change how sepsis is diagnosed and treated. Their work suggests that two different types of immune cells are involved in the progression of sepsis: one cell type that's protective and another that's harmful. These findings could help doctors better identify who is most at risk for severe complications from the disease. "The results from our study show that there are two distinct populations of monocytes," said co-author Dr. Michael Jaffe, professor of medicine at Washington University. "One population promotes inflammation and is associated with worse outcomes; the other helps resolve inflammation and promotes healing." Monocytes are a type of white blood cell that are part of the innate immune system. They play a key role in fighting off bacteria and viruses. When someone has sepsis, their monocytes become activated. That means they increase in number and start producing cytokines, which are small proteins that regulate the body's immune response. Typically, when the body encounters an infection, these cytokines signal the immune system to fight back. But too many cytokines produced too quickly can result in a hyperinflammatory state, which can lead to organ failure and death. In the past, doctors have used blood tests to diagnose sepsis. However, those tests only measure the overall levels of cytokines in a person's blood. The new study suggests that it might be possible to determine whether someone has a particular type of monocyte that could make them more susceptible to sepsis. If so, doctors could potentially use this information to personalize treatments and improve patient outcomes. "Our findings suggest that there may be opportunities to develop therapies that target specific monocyte subsets," said co-author Dr. Daniel Levy, assistant professor of medicine at Washington University. "This could potentially reduce the risk of developing sepsis or even improve outcomes once someone already has sepsis." There's still much to learn about how these monocyte subsets contribute to sepsis development and progression, but the results of this study provide promising insights into why some patients respond differently to treatment than others.

Table 5.29: Example response of Mistral-v3-7b (base).

5.8 Conclusion

We demonstrate the presence of idiosyncrasies in Large Language Models (LLMs) and investigate a synthetic task designed to quantify their extent. We find that simply fine-tuning pretrained text embedding models on LLM outputs leads to exceedingly high accuracy in predicting the origins of the text. This phenomenon persists across diverse prompt datasets, LLM combinations, and many other settings. We also pinpoint concrete forms of these idiosyncrasies within LLMs. We hope our work encourages further research into understanding idiosyncrasies in LLMs.

Chapter 6

Conclusion

In this dissertation, we conducted a thorough and systematic study of Large Language Models (LLMs), despite their inherently complex and opaque nature. Through this work, we uncovered several important hidden properties of LLMs and revealed the internal mechanisms behind these properties. These findings significantly advance our understanding of LLMs and offer valuable insights for the development of future models.

The weight space is a fundamental component of deep learning models, with modern LLMs often comprising billions of parameters. In Chapter 3, we demonstrate the intrinsic sparsity of the LLM weight space. We examine the counterintuitive phenomenon that the popular magnitude pruning method fails drastically on LLMs. We find that magnitude pruning overlooks emergent activation outliers common in LLMs. Motivated by this, we propose Wanda, a simple pruning approach that jointly considers weight magnitudes and input activations. We use Wanda to find efficient and sparse subnetworks from pretrained LLMs, without inducing any changes to the preserved weights.

Unlike static weight parameters, activations are dynamic, and LLMs use these internal representations to encode input data. In Chapter 4, we reveal the widespread presence of massive activations – structured outliers in the activation space of LLMs. We conduct a comprehensive analysis of these activations. Through intervention analysis, we show these activations function as crucial, input-agnostic biases. Furthermore, we find that they are closely connected to the attention concentration patterns, and we propose an alternative attention formulation that augments self-attention with learnable key and value embeddings. We show that pretraining with this modified attention mechanism eliminates massive activations.

Finally, LLMs are interesting because of their ability to generate natural language outputs. In Chapter 5 of the thesis, we examine the output behaviors of LLMs, where we are mainly motivated by understanding the differences between models beyond benchmark performance. We develop a conceptually simple framework to evaluate idiosyncrasies in LLM generations – a synthetic classification task with a goal of predicting the origin of a given output. Using this framework, we find that outputs from different LLMs are highly distinguishable: they can be distinguished with extremely high accuracies. We further identify concrete forms of these idiosyncrasies , including word-level distributions, formatting style and high-level semantics. Finally, we discuss how our findings have implications for the use of synthetic data, model similarity inference, and the design of more reliable evaluation protocols.

While this dissertation is centered on large language models (LLMs), the problems investigated and methodologies used span a diverse range of topics, including sparsity, pruning, quantization, deep learning architectures, and linguistics. By examining these topics through the lens of LLMs, the insights and approaches developed in this thesis provide a solid foundation for future research efforts aimed at understanding advanced deep learning models. This will be increasingly relevant as the field continues to evolve.

6.1 Future Directions

We conclude this thesis by discussing several potential future directions based on our findings.

Sparsity and efficiency Researchers have long studied sparsity in neural networks. However, sparse neural network face a practicle challenge where unstructured sparsity is not natively supported by modern GPU hardware. This leads to a significant gap between theoretical and practical efficiency. Therefore an important question is how to bridge this gap and unlock the potential of sparsity for accelerating LLMs. One promising direction is via a special type of sparsity, called structured sparsity. Structured sparsity [139] is a type of fine-grained weight sparsity that is supported by latest NVIDIA GPUs. However, as we have shown in Chapter 3, structured sparse LLMs are not as effective as unstructured sparse ones. Moreover, there still remains a large performance gap between structured sparse models and their dense counterparts. A natural question is whether we can leverage the insights from our work to design more effective pruning methods for structured sparsity.

Another promising direction is to explore the sparse property beyond weight parameters. Some of the questions we can ask are: are the activations of LLMs also sparse? If so, can we leverage the activation sparsity to make the training or inference of LLMs more efficient? Moreover, this analysis can be extended to the gradients during training. If gradients are sparse, we can significantly reduce the communication cost in distributed training, which is highly desirable as we continue to scale up LLMs.

Low-precision training In addition to sparsity, quantization is another important direction for improving the efficiency of deep learning models. Unlike sparsity, which is less explored in the context of LLMs, quantization has been widely studied and adopted, especially for model development. However, most of the existing methods rely on post-training quantization, where a pretrained full-precision model is quantized after training. Low-precision training has been less explored, due to the training instability that arises from low-precision weights and activations. Our findings in Chapter 4 reveal one such bottleneck: the existence of extreme activation outliers.

This raises a natural question: how can we better understand the training dynamics under low-precision regimes, and can we develop methods to improve their stability? At the moment, we are seeing 8-bit floating-point (8-bit FP) as the most common low-precision format for pretraining LLMs [6]. In the future, we may see even lower precision formats, such as 4-bit floating point or even binary representations. In these extereme quantization settings, what are the fundamental challenges, and how can we design effective mitigation strategies? We hope our work can provide some valuable insights on these directions.

Evaluating frontier models We are living in an era where frontier LLMs are being developed at a rapid race, with frequent model releases and steady increase in benchmark performance. However, this raises a key challenge: how should we meaningfully evaluate progress, especially as benchmarks can be saturated over time. Moreover, recent studies highlight limitations and potential flaws in popular benchmarks [28]. Our work in Chapter 5 explores one aspect of this question, where we propose a framework to evaluate and understand model differences. Going forward, how do we capture these differences in a scalable way remains a question, given that reasoning models have much longer outputs and agentic systems can have long trajectories. Ultimately, the research community needs to find fair and reliable methods to evaluate these advanced models, which is important for guiding the future development of frontier models.

Bibliography

[1] Ashish Vaswani, Noam Sharzeer, Niki, Parmar, et al. Attention Is All You Need. *NeurIPS*, 2017. 1, 2.2, 5.3

[2] Alec Radford, Jeffrey Wu, Rewon Child, et al. Language Models are Unsupervised Multitask Learners. *Technical Report*, 2019. 1, 2.3, 4.5.2, 5.3, 5.3.2, 5.7.3

[3] Tom B Brown, Benjamin, Mann, Nick, Ryder, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020. 1, 3.2, 4.2

[4] OpenAI. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*, 2023. 1, 3.2, 4.2

[5] OpenAI. Introducing ChatGPT. *OpenAI Website*, 2022. 1

[6] DeepSeek-AI. DeepSeek-V3 Technical Report. *arXiv preprint arXiv:2412.19437*, 2024. [cite: 298] 1, 1, 6.1

[7] OpenAI. Sora. *OpenAI Website*, 2024. 1

[8] Alec Radford, Jong Wook Kim, Tao Xu, et al. Robust Speech Recognition via Large-Scale Weak Supervision. *arXiv preprint arXiv:2212.04356*, 2022. 1

[9] Jason Wei, Xuezhi Wang, Dale, Schuurmans, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. *Advances in Neural Information Processing Systems*, 2022. 1, 4.2

[10] Edward J. Hu, Yelong Shen, Phillip Wallis, et al. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021. 1, 3.6, 5.7.3

[11] Qingxiu Dong, Lei Li, Damai Dai, et al. A Survey on In-context Learning. *arXiv preprint arXiv:2301.00234*, 2022. 1

[12] Carlos E. Jimenez, John Yang, Alexander Wettig, et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? *arXiv preprint arXiv:2310.06770*, 2024. 1

[13] OpenAI. GPT-4o System Card. *arXiv preprint arXiv:2410.21276*, 2024. [cite: 291] 2.1, 1

[14] Jin Xu, Zhifang Guo, Jinzheng He, et al. Qwen2.5-Omni Technical Report. *arXiv preprint arXiv:2503.20215*, 2025. 2.1

[15] Anthropic. Claude-3.5-Sonnet. `https://www.anthropic.com/news/claude-3-5-sonnet`, 2024. [cite: 292] 1

[16] xAI. Grok-2 Beta Release. `https://x.ai/blog/grok-2`, 2024. [cite: 292] 1

[17] Albert Gu and Tri Dao Mamba: Linear-Time Sequence Modeling with Selective State Spaces. *arXiv preprint arXiv:2312.00752*, 2023. 2.2

[18] Jacob Devlin, Ming-Wei Chang, Kenton Lee, et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint arXiv:1810.04805*, 2018. 2.2, 2.3, 3.8.3, 4.7, 5.3.2, 5.4

[19] Tri Dao and Albert Gu Transformers are SSMs: Generalized Models and Efficient Algorithms Through Structured State Space Duality. *arXiv preprint arXiv:2405.21060*, 2024. 2.2

[20] Shen Nie, Fengqi Zhu, Zebin You, et al. Large Language Diffusion Models. *arXiv preprint arXiv:2502.09992*, 2025. 2.2

[21] Subham Sekhar Sahoo, Marianne Arriola, Yair Schiff, et al. Simple and Effective Masked Diffusion Language Models. *arXiv preprint arXiv:2406.07524*, 2024. 2.2

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, et al. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023. 2.2, 3.2, 3.5, 3.8.7

[23] Hugo, Touvron, Louis Martin, Kevin Stone, et al. Llama 2: Open Foundation and Fine-Tuned Chat Models. *arXiv preprint arXiv:2307.09288*, 2023. 2.2, 3.2, 3.5, 4.2

[24] Niklas Muennighoff, Zitong Yang, Weijia Shi, et al. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025. 1

[25] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, et al. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*, 2024. 2.2, 4.8.1, 2

[26] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022. 2.3, 4.8.2

[27] Noam Shazeer. GLU Variants Improve Transformer. *arXiv preprint arXiv:2002.05202*, 2020. 2.2

[28] Shivalika Singh, Yiyang Nan, Alex Wang, et al. The Leaderboard Illusion. *arXiv preprint arXiv:2504.20879*, 2025. 5.5, 6.1

[29] Yuntao Bai, Saurav Kadavath, Sandipan Kundu, et al. Constitutional AI: Harmlessness from AI Feedback. *arXiv preprint arXiv:2212.08073*, 2022. 2.3

[30] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. Deep Residual Learning for Image Recognition. *CVPR*, 2016. 2.2, 4.3

[31] Jimmy Lei Ba, Jamie Ryan Kiros, Geoffrey E. Hinton. Layer Normalization. *arXiv preprint arXiv:1607.06450*, 2016. (document), 2.2, 4.5.2, 4.8.6, 4.37

[32] Long Ouyang, Jeff Wu, Xu Jiang, et al. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[33] Pranab Sahoo, Ayush Kumar Singh, Sriparna Saha, et al. A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications. *arXiv preprint arXiv:2402.07927*, 2024. 1

[34] Chen Liang, Simiao Zuo, Minshuo Chen, et al. Super tickets in pre-trained language models: From model compression to improving generalization. *arXiv preprint arXiv:2105.12002*, 2021.

[35] Jason Wei, Karina Nguyen, Hyung Won Chung, et al. Measuring Short-Form Factuality in

Large Language Models. *arXiv preprint arXiv:2411.04368*, 2024. 1

[36] Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 2020. 2.2, 3.5, 4.8.12

[37] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv preprint arXiv:1907.11692*, 2019. 2.2

[38] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv preprint arXiv:1409.0473*, 2014. 2.2

[39] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long Short-Term Memory-Networks for Machine Reading. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2016. 2.2

[40] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, et al. A Structured Self-attentive Sentence Embedding. *arXiv preprint arXiv:1703.03130*, 2017. 2.2

[41] Rico Sennrich, Barry Haddow, Alexandra Birch, et al. Neural Machine Translation of Rare Words with Subword Units. *arXiv preprint arXiv:1508.07909*, 2015. 2.2

[42] Dan Hendrycks and Kevin Gimpel. Gaussian Error Linear Units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016. 2.2

[43] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for Activation Functions. *arXiv preprint arXiv:1710.05941*, 2017. 2.2

[44] Leo Gao, Stella Biderman, Sid Black, et al. The Pile: An 800GB Dataset of Diverse Text for Language Modeling. *arXiv preprint arXiv:2101.00027*, 2021. 4.8.8, 4.8.12

[45] Together Computer. RedPajama: An Open Source Recipe to Reproduce LLaMA training dataset. *https://github.com/togethercomputer/RedPajama-Data*, 2023. 4.3, 4.8.12

[46] Abhimanyu Dubey, Moitreya Chatterjee, Narendra Ahuja. Coreset-Based Neural Network Compression. *ECCV*, 2018. 3.7, 3.8.3

[47] Mohammad Babaeizadeh, Paris Smaragdis, Roy H. Campbell. NoiseOut: A Simple Way to Prune Neural Networks. *arXiv preprint arXiv:1611.06211*, 2016. 3.7, 3.8.3

[48] Hengyuan Hu, Rui Peng, Yu-Wing Tai, et al. Network Trimming: A Data-Driven Neuron Pruning Approach towards Efficient Deep Architectures. *arXiv preprint arXiv:1607.03250*, 2016. 3.7

[49] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, et al. Stochastic Activation Pruning for Robust Adversarial Defense. *International Conference on Learning Representations*, 2018. 3.7

[50] Gongfan Fang, Xinyin Ma, Mingli Song, et al. DepGraph: Towards Any Structural Pruning. *Conference on Computer Vision and Pattern Recognition*, 2023. 3.7

[51] Pavlo Molchanov, Arun Mallya, Stephen Tyree, et al. Importance Estimation for Neural Network Pruning. *CVPR*, 2019. 3.7

[52] Leo Gao, Jonathan Tow, Stella Biderman, et al. A framework for few-shot language model

evaluation. *Version v0.0.1. Sept*, 2021. 3.5

[53] Percy Liang, Rishi Bommasani, Tony Lee, et al. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

[54] Yefan Zhou, Yaoqing Yang, Arin Chang, et al. A Three-regime Model of Network Pruning. *ICML*, 2023. 3.7

[55] Azade Nova, Hanjun Dai, Dale Schuurmans. Gradient-Free Structured Pruning with Unlabeled Data. *International Conference on Machine Learning*, 2023. 3.7

[56] Zichang Liu, Jue Wang, Tri Dao, et al. Deja Vu: Contextual Sparsity for Efficient LLMs at Inference Time. *ICML*, 2023. 3.7

[57] Guangxuan Xiao, Ji Lin, Mickael Seznec, et al. SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models. *ICML*, 2023. 3.2, 3.5, 3.7, 4.7, 4.9

[58] Tim Dettmers, Mike Lewis, Younes Belkada, et al. LLM.int8(): 8-bit Matrix Multiplication for Transformers at Scale. *NeurIPS*, 2022. 3.2, 3.2, 3.3, 3.4, 3.7, 4.2, 4.3.3, 4.7, 4.9

[59] Elias Frantar, Dan Alistarh. SparseGPT: Massive Language Models Can Be Accurately Pruned in One-Shot. *ICML*, 2023. 3.2, 3.2, 3.4, 3.5, 3.5, 3.5.3, 3.7, 3.8.4, 3.8.6

[60] Elena Voita, Javier Ferrando, Christoforos Nalmpantis. Neurons in Large Language Models: Dead, N-gram, Positional. *arXiv preprint arXiv:2309.04827*, 2023. 3.7

[61] Mengzhou Xia, Mikel Artetxe, Chunting Zhou, et al. Training Trajectories of Language Models Across Scales. *arXiv preprint arXiv:2212.09803*, 2022.

[62] Mitch Marcus, Grace Kim, Mary Ann Marcinkiewicz, et al. The Penn treebank: Annotating predicate argument structure. *Human Language Technology: Proceedings of a Workshop held at Plainsboro, New Jersey, March 8-11, 1994*, 1994.

[63] Stephen Merity, Caiming Xiong, James Bradbury, et al. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016. 3.5, 4.8.12

[64] Susan Zhang, Stephen Roller, Naman Goyal, et al. OPT: Open pre-trained transformer language models. *arXiv preprint arXiv:2205.01068*, 2022. 3.7, 3.8.2, 3.8.2, 3.8.4

[65] Teven Le Scao, Angela Fan, Christopher Akiki, et al. Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*, 2022. 3.8.2, 3.8.2

[66] Song Han, Jeff Pool, John Tran, et al. Learning both weights and connections for efficient neural networks. *NeurIPS*, 2015. 3.2, 3.2, 3.3, 3.5, 3.7

[67] Trevor Gale, Erich Elsen, Sara Hooker. The state of sparsity in deep neural networks. *ICML*, 2019. 3.2, 3.7, 3.8.1

[68] Victor Sanh, Thomas Wolf, Alexander M. Rush. Movement Pruning: Adaptive Sparsity by Fine-Tuning. *NeurIPS*, 2020. 3.7, 3.8.3

[69] Jonathan Frankle, Michael Carbin. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. *ICLR*, 2019. 3.2, 3.3

[70] Michael Zhu, Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017. 3.2

[71] Zhuohan Li, Eric Wallace, Sheng Shen, et al. Train Large, Then Compress: Rethinking Model Size for Efficient Training and Inference of Transformers. *International Conference on Machine Learning*, 2020.

[72] Aojun Zhou, Yukun Ma, Junnan Zhu, et al. Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch. *International Conference on Learning Representations*, 2021.

[73] Itay Hubara, Brian Chmiel, Moshe Island, et al. Accelerated Sparse Neural Training: A Provable and Efficient Method to find N:M Transposable Masks. *NeurIPS*, 2021. 3.7

[74] Zhuang Liu, Mingjie Sun, Tinghui Zhou, et al. Rethinking the Value of Network Pruning. *ICLR*, 2019. 3.2, 3.7

[75] Elias Frantar, Sidak Pal Singh, Dan Alistarh. Optimal Brain Compression: A Framework for Accurate Post-Training Quantization and Pruning. *NeurIPS*, 2022. 3.7

[76] Elias Frantar, Dan Alistarh. SPDY: Accurate Pruning with Speedup Guarantees. *ICML*, 2022. 3.7

[77] Yann LeCun, John S. Denker, Sara A. Solla. Optimal brain damage. *NeurIPS*, 1989. 3.2, 3.4, 3.7

[78] Babak Hassibi, David G. Stork, Gregory J. Wolff. Optimal brain surgeon and general network pruning. *IEEE International Conference on Neural Networks*, 1993. 3.2, 3.4, 3.7

[79] Christos Louizos, Max Welling, Diederik P. Kingma. Learning Sparse Neural Networks through L0 Regularization. *ICLR*, 2018. 3.2

[80] Alex Renda, Jonathan Frankle, Michael Carbin. Comparing Rewinding and Fine-tuning in Neural Network Pruning. *ICLR*, 2020. 3.2, 3.7

[81] Song Han, Huizi Mao, William J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. *ICLR*, 2016. 3.7

[82] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M. Roy, et al. Stabilizing the Lottery Ticket Hypothesis. *ICML*, 2020. 3.7

[83] Woosuk Kwon, Sehoon Kim, Michael W. Mahoney, et al. A Fast Post-Training Pruning Framework for Transformers. *NeurIPS*, 2022. 3.7

[84] Sidak Pal Singh, Dan Alistarh. WoodFisher: Efficient Second-Order Approximation for Neural Network Compression. *NeurIPS*, 2020. 3.7

[85] Aditya Kusupati, Vivek Ramanujan, Raghav Somani, et al. Soft threshold weight reparameterization for learnable sparsity. *ICML*, 2020. 3.7

[86] Jason Wei, Yi Tay, Rishi Bommasani, et al. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research*, 2022. 3.2, 3.7

[87] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, et al. GPTQ: Accurate Post-training Compression for Generative Pretrained Transformers. *ICLR*, 2023. 3.2

[88] Dan Hendrycks, Collin Burns, Steven Basart, et al. Measuring massive multitask language understanding. *ICLR*, 2021. 3.8.7, 4.5.1, 4.8.12

[89] Rylan Schaeffer, Brando Miranda, Sanmi Koyejo. Are Emergent Abilities of Large

Language Models a Mirage? *arXiv preprint arXiv:2304.15004*, 2023. 3.7

[90] Olga Kovaleva, Saurabh Kulshreshtha, Anna Rogers, et al. BERT Busters: Outlier Dimensions that Disrupt Transformers. *ACL Findings*, 2021. 3.7, 4.7

[91] Hritik Bansal, Karthik Gopalakrishnan, Saket Dingliwal, et al. Rethinking the Role of Scale for In-Context Learning: An Interpretability-based Case Study at 66 Billion Scale. *Association for Computational Linguistics (ACL)*, 2023. 3.7

[92] Siyu Ren, Kenny Q. Zhu. Pruning Pre-trained Language Models with Principled Importance and Self-regularization. *ACL*, 2023. 3.8.3

[93] Mengzhou Xia, Zexuan Zhong, Danqi Chen. Structured Pruning Learns Compact and Accurate Models. *Association for Computational Linguistics (ACL)*, 2022. 3.7

[94] Zhuang Liu, Jianguo Li, Zhiqiang Shen, et al. Learning Efficient Convolutional Networks through Network Slimming. *ICCV*, 2017. 3.7

[95] Davis Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, et al. What is the State of Neural Network Pruning? *Proceedings of Machine Learning and Systems*, 2020. 3.2, 3.3, 3.8.1

[96] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, et al. The Falcon Series of Open Language Models. *arXiv preprint arXiv:2102.00554*, 2021.

[97] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, et al. Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks. *arXiv preprint arXiv:2102.00554*, 2021.

[98] Christopher Clark, Kenton Lee, Ming-Wei Chang, et al. BoolQ: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*, 2019. 3.8.9, 4.8.12

[99] Alex Wang, Amanpreet Singh, Julian Michael, et al. Glue: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv preprint arXiv:1804.07461*, 2018. 3.8.9

[100] Rowan Zellers, Ari Holtzman, Yonatan Bisk, et al. HellaSwag: Can a Machine Really Finish Your Sentence? *arXiv preprint arXiv:1905.07830*, 2019. 3.8.9

[101] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, et al. WinoGrande: An Adversarial Winograd Schema Challenge at Scale. *arXiv preprint arXiv:1907.10641*, 2019. 3.8.9, 4.8.12

[102] Peter Clark, Isaac Cowhey, Oren Etzioni, et al. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018. 3.8.9, 4.8.12

[103] Todor Mihaylov, Peter Clark, Tushar Khot, et al. Can a suit of armor conduct electricity? a new dataset for open book question answering. *arXiv preprint arXiv:1809.02789*, 2018. 3.8.9

[104] Eldar Kurtic, Daniel Campos, Tuan Nguyen, et al. The Optimal BERT Surgeon: Scalable and Accurate Second-Order Pruning for Large Language Models. *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2022.

[105] Michael Bommarito, Daniel Martin Katz. GPT Takes the Bar Exam. *arXiv preprint*

arXiv:2212.14402, 2022. 3.2, 4.2

[106] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, et al. Training Compute-Optimal Large Language Models. *arXiv preprint arXiv:2203.15556*, 2022. 3.7

[107] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, et al. Sparks of Artificial General Intelligence: Early experiments with GPT-4. *arXiv preprint arXiv:2303.12712*, 2023. 3.2, 4.2

[108] . . `https://github.com/ggerganov/llama.cpp`, 2023.

[109] Zhewei Yao, Reza Yazdani Aminabadi, Minjia Zhang, et al. ZeroQuant: Efficient and Affordable Post-Training Quantization for Large-Scale Transformers. *Advances in Neural Information Processing Systems*, 2022.

[110] Tim Dettmers, Luke Zettlemoyer. The case for 4-bit precision: k-bit Inference Scaling Laws. *arXiv preprint arXiv:2212.09720*, 2022.

[111] Ying Sheng, Lianmin Zheng, Binhang Yuan, et al. High-throughput Generative Inference of Large Language Models with a Single GPU. *ICML*, 2023. 3.2

[112] Xinyin Ma, Gongfan Fang, Xinchao Wang. LLM-Pruner: On the Structural Pruning of Large Language Models. *arXiv preprint arXiv:2305.11627*, 2023. 3.7

[113] Song Han, Jeff Pool, Sharan Narang, et al. DSD: Dense-Sparse-Dense Training for Deep Neural Networks. *International Conference on Learning Representations*, 2017.

[114] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel Roy, et al. Linear Mode Connectivity and the Lottery Ticket Hypothesis. *ICML*, 2020.

[115] Yucheng Lu, Shivani Agrawal, Suvinay Subramanian, et al. STEP: Learning N:M Structured Sparsity Masks from Scratch with Precondition. *ICML*, 2023.

[116] Advait Gadhikar, Sohom Mukherjee, Rebekka Burkholz. Why Random Pruning Is All We Need to Start Sparse. *ICML*, 2023. 3.7

[117] Duc Hoang, Shiwei Liu, Radu Marculescu, et al. Revisiting pruning at initialization through the lens of ramanujan graph? *ICLR*, 2023. 3.7

[118] Mansheej Paul, Feng Chen, Brett W. Larsen, et al. Unmasking the Lottery Ticket Hypothesis: What's Encoded in a Winning Ticket's Mask? *ICLR*, 2023. 3.7

[119] Pavlo Molchanov, Stephen Tyree, Tero Karras, et al. Pruning Convolutional Neural Networks for Resource Efficient Inference. *ICLR*, 2017. 3.7, 3.8.3

[120] Xiuying Wei, Yunchen Zhang, Xiangguo Zhang, et al. Outlier Suppression: Pushing the Limit of Low-bit Transformer Language Models. *NeurIPS*, 2022. 3.7

[121] Corinna Cortes, Mehryar Mohri, Afshin Rostamizadeh. L2 Regularization for Learning Kernels. *Conference on Uncertainty in Artificial Intelligence*, 2009.

[122] Aitor Lewkowycz, Guy Gur-Ari. On the training dynamics of deep networks with L2 regularization. *Advances in Neural Information Processing Systems*, 2020.

[123] Shiwei Liu, Tianlong Chen, Zhenyu Zhang, et al. Sparsity May Cry: Let Us Fail (Current) Sparse Neural Networks Together! *ICLR*, 2023. 3.7

[124] Tian Jin, Michael Carbin, Daniel M. Roy, et al. Pruning's Effect on Generalization Through the Lens of Training and Regularization. *NeurIPS*, 2022. 3.5.3

[125] Maying Shen, Hongxu Yin, Pavlo Molchanov, et al. Structural pruning via latency-saliency knapsack. *NeurIPS*, 2022. 3.7

[126] . . `https://en.wikipedia.org/wiki/Knapsack_problem`, 2023.

[127] Jia Deng, Wei Dong, Richard Socher, et al. ImageNet: A large-scale hierarchical image database. *CVPR*, 2009. 3.8.1, 4.8.12

[128] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, et al. A ConvNet for the 2020s. *CVPR*, 2022. 3.8.1

[129] Namhoon Lee, Thalaiyasingam Ajanthan, Philip H. S. Torr. SNIP: Single-shot Network Pruning based on Connection Sensitivity. *ICLR*, 2018. 3.8.3

[130] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *ICLR*, 2021. 3.8.1, 4.6

[131] Qingru Zhang, Simiao Zuo, Chen Liang, et al. PLATON: Pruning Large Transformer Models with Upper Confidence Bound of Weight Importance. In *International Conference on Machine Learning*, 2021. 3.8.3

[132] Hugo Touvron, Matthieu Cord, Matthijs Douze, et al. Training data-efficient image transformers & distillation through attention. *International Conference on Machine Learning*, 2021.

[133] Ze Liu, Yutong Lin, Yue Cao, et al. Swin transformer: Hierarchical vision transformer using shifted windows. *International Conference on Computer Vision*, 2021.

[134] Edward J. Hu, Yelong Shen, Phillip Wallis, et al. LoRA: Low-Rank Adaptation of Large Language Models. *ICLR*, 2021. 1, 3.6, 5.7.3

[135] William Timkey, Marten van Schijndel. All Bark and No Bite: Rogue Dimensions in Transformer Language Models Obscure Representational Quality. *arXiv:2109.04404*, 2021. 3.7, 4.7

[136] Yelysei Bondarenko, Markus Nagel, Tijmen Blankevoort. Understanding and Overcoming the Challenges of Efficient Transformer Quantization. *arXiv:2109.12948*, 2021. 3.7, 4.7

[137] Ziyang Luo, Artur Kulmizev, Xiaoxi Mao. Positional Artefacts Propagate Through Masked Language Model Embeddings. *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, 2021. 3.7

[138] Giovanni Puccetti, Anna Rogers, Aleksandr Drozd, et al. Outliers Dimensions that Disrupt Transformers Are Driven by Frequency. *arXiv preprint arXiv:2205.11380*, 2022. 3.7

[139] Asit Mishra, Jorge Albericio Latorre, Jeff Pool, et al. Accelerating Sparse Deep Neural Networks. *arXiv preprint arXiv:2104.08378*, 2021. 3.4, 6.1

[140] Jason Wei, Najoung Kim, Yi Tay, et al. Inverse scaling can become U-shaped. *arXiv preprint arXiv:2211.02011*, 2022.

[141] Stella Biderman, Hailey Schoelkopf, Quentin Anthony, et al. Pythia: A Suite for Analyzing

Large Language Models Across Training and Scaling. *arXiv preprint arXiv:2304.01373*, 2023. 3.8.2

[142] Angela Fan, Edouard Grave, Armand Joulin. Reducing Transformer Depth on Demand with Structured Dropout. *International Conference on Learning Representations*, 2020. 3.7

[143] Chong Yu, Tao Chen, Zhongxue Gan, et al. Boost Vision Transformer with GPU-Friendly Sparsity and Quantization. *CVPR*, 2023.

[144] Tianlong Chen, Jonathan Frankle, Shiyu Chang, et al. The Lottery Ticket Hypothesis for Pre-trained BERT Networks. *NeurIPS*, 2020. 3.8.3

[145] Jeff Pool, Chong Yu. Channel Permutations for N:M Sparsity. *NeurIPS*, 2021.

[146] Feiwen Zhu, Jeff Pool, Michael Andersch, et al. Sparse Persistent RNNs: Squeezing Large Recurrent Networks On-Chip. *International Conference on Learning Representations*, 2018.

[147] Kayhan Behdin, Ayan Archarya, Aman Gupta, et al. QuantEase: Optimization-based Quantization for Language Models – An Efficient and Intuitive Algorithm. *arXiv preprint arXiv:2309.01885*, 2023. 3.7

[148] Elias Frantar, Carlos Riquelme, Neil Houlsby, et al. Scaling Laws for Sparsely-Connected Foundation Models. *arXiv preprint arXiv:2309.08520*, 2023. 3.9

[149] Alexandra Peste, Eugenia Iofinova, Adrian Vladu, et al. AC/DC: Alternating Compressed/DeCompressed Training of Deep Neural Networks. *NeurIPS*, 2021. 3.9

[150] Ji Lin, Jiaming Tang, Haotian Tang, et al. AWQ: Activation-aware Weight Quantization for LLM Compression and Acceleration. *arXiv preprint arXiv:2306.00978*, 2023. 3.2, 3.7, 4.7

[151] Riade Benbaki, Wenyu Chen, Xiang Meng, et al. Fast as CHITA: Neural Network Pruning with Combinatorial Optimization. *ICML*, 2023. 3.9

[152] Denis Kuznedelev, Eldar Kurtic, Eugenia Iofinova, et al. Accurate Neural Network Pruning Requires Rethinking Sparse Optimization. *arXiv preprint arXiv:2308.02060*, 2023. 3.9

[153] Utku Evci, Trevor Gale, Jacob Menick, et al. Rigging the Lottery: Making All Tickets Winners. *ICML*, 2020. 3.5.3, 3.9

[154] Arash Ahmadian, Saurabh Dash, Hongyu Chen, et al. Intriguing Properties of Quantization at Scale. *NeurIPS*, 2023. 3.2, 4.7

[155] Tim Dettmers, Ruslan Svirschevski, Vage Egiazarian, et al. SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression. *arXiv preprint arXiv:2306.03078*, 2023. 3.7

[156] Xin Dong, Shangyu Chen, Sinno Jialin Pan. Learning to Prune Deep Neural Networks via Layer-wise Optimal Brain Surgeon. *Advances in Neural Information Processing Systems*, 2017.

[157] Alireza Aghasi, Afshin Abdi, Nam Nguyen, et al. Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee. *arXiv preprint arXiv:1611.05162*, 2016.

[158] Dirk Groeneveld, Iz Beltagy, Pete Walsh, et al. OLMo: Accelerating the Science of Language Models. *arXiv preprint arXiv:2402.00838*, 2024.

[159] Ebtesam Almazrouei, Hamza Alobeidli, Abdulaziz Alshamsi, et al. The Falcon Series of Open Language Models. *arXiv preprint arXiv:2311.16867*, 2023. 4.8.1

[160] MosaicML. Introducing MPT-7B: A New Standard for Open-Source, Commercially Usable LLMs. *www.mosaicml.com/blog/mpt-7b*, 2023. 4.8.1

[161] Mojan Javaheripi, Sébastien Bubeck, et al. Phi-2: The surprising power of small language models. `https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/`, 2023. 4.3.1

[162] Jan Leike, Jeffrey Wu, Steven Bills, et al. Language models can explain neurons in language models. `https://openai.com/research/language-models-can-explain-neurons-in-language-models`, 2023.

[163] Stefan Heimersheim, Alex Turner. Residual stream norms grow exponentially over the forward pass. `https://www.alignmentforum.org/posts/8mizBCm3dyc432nK8/residual-stream-norms-grow-exponentially-over-the-forward`, 2023. 4.7

[164] Evan Miller. Attention Is Off By One. `https://www.evanmiller.org/attention-is-off-by-one.html`, 2023. 4.7, 1

[165] Andrej Karpathy. NanoGPT. `https://github.com/karpathy/nanoGPT`, 2023. 4.8.8

[166] Wes Gurnee, Theo Horsley, Zifan Carl Guo, et al. Universal Neurons in GPT2 Language Models. *arXiv preprint arXiv:2401.12181*, 2024.

[167] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, et al. Small-scale proxies for large-scale Transformer training instabilities. *arXiv preprint arXiv:2309.14322*, 2023. 4.5.3, 4.9

[168] Rafael Rafailov, Archit Sharma, Eric Mitchell, et al. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. *arXiv preprint arXiv:2305.18290*, 2023.

[169] Chengrun Yang, Xuezhi Wang, Yifeng Lu, et al. Large Language Models as Optimizers. *arXiv preprint arXiv:2309.03409*, 2023. 4.2

[170] Neel Nanda, Lawrence Chan, Tom Lieberum, et al. Progress measures for grokking via mechanistic interpretability. *International Conference on Learning Representations*, 2023.

[171] Andy Zou, Zifan Wang, Nicholas Carlini, et al. Universal and Transferable Adversarial Attacks on Aligned Language Models. *arXiv preprint arXiv:2307.15043*, 2023.

[172] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mixtral of Experts. *arXiv preprint arXiv:2401.04088*, 2024. 4.3

[173] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, et al. Mistral 7B. *arXiv preprint arXiv:2310.06825*, 2023. 4.8.1, 2

[174] Mark Chen, Jerry Tworek, Heewoo Jun, et al. Evaluating Large Language Models Trained on Code. *arXiv preprint arXiv:2107.03374*, 2021.

[175] Wes Gurnee, Neel Nanda, Matthew Pauly, et al. Finding Neurons in a Haystack: Case

Studies with Sparse Probing. *arXiv preprint arXiv:2305.01610*, 2023.

[176] Biao Zhang, Rico Sennrich. Root Mean Square Layer Normalization. *NeurIPS*, 2019. (document), 1, 4.8.6, 4.37

[177] Nelson Elhage, Neel Nanda, Catherine Olsson, et al. A Mathematical Framework for Transformer Circuits. *Transformer Circuits Thread*, 2021.

[178] Nora Belrose, Zach Furman, Logan Smith, et al. Eliciting Latent Predictions from Transformers with the Tuned Lens. *arXiv preprint arXiv:2303.08112*, 2023.

[179] Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context Learning and Induction Heads. *arXiv preprint arXiv:2209.11895*, 2022.

[180] Jianlin Su, Yu Lu, Shengfeng Pan, et al. RoFormer: Enhanced Transformer with Rotary Position Embedding. *arXiv preprint arXiv:2104.09864*, 2021. 4.7

[181] Ofir Press, Noah A. Smith, Mike Lewis. Train Short, Test Long: Attention with Linear Biases Enables Input Length Extrapolation. *arXiv preprint arXiv:2108.12409*, 2021. 4.7

[182] Mahdi Namazifar, Devamanyu Hazarika, Dilek Hakkani-Tur. Role of Bias Terms in Dot-Product Attention. *arXiv preprint arXiv:2302.08626*, 2023. 4.7

[183] Brian S. Robinson, Nathan Drenkow, Colin Conwell, et al. A sparse null code emerges in deep neural networks. *NeurIPS UniReps Workshop*, 2023. 4.7

[184] Aohan Zeng, Xiao Liu, Zhengxiao Du, et al. GLM-130B: An Open Bilingual Pre-trained Model. *arXiv preprint arXiv:2210.02414*, 2022. 4.7

[185] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, et al. Compressive Transformers for Long-Range Sequence Modelling. *arXiv preprint*, 2019. 4.8.12

[186] Catherine Olsson, Nelson Elhage, Neel Nanda, et al. In-context Learning and Induction Heads. *Transformer Circuits Thread*, 2022. 4.9

[187] Jerry Yao-Chieh Hu, Pei-Hsuan Chang, Robin Luo, et al. Outlier-Efficient Hopfield Layers for Large Transformer-Based Models. *arXiv preprint arXiv:2404.03828*, 2024. 4.7

[188] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, et al. PIQA: Reasoning about Physical Commonsense in Natural Language. *arXiv preprint arXiv:1911.11641*, 2019. 4.8.12

[189] Jun Zhao, Zhihao Zhang, Yide Ma, et al. Unveiling A Core Linguistic Region in Large Language Models. *arXiv:2310.14928*, 2023. 4.7

[190] Guangxuan Xiao, Yuandong Tian, Beidi Chen, et al. Efficient Streaming Language Models with Attention Sinks. *arXiv:2309.17453*, 2023. 4.2, 4.5.1, 4.5.3, 4.7, 4.8.8

[191] Timothée Darcet, Maxime Oquab, Julien Mairal, et al. Vision Transformers Need Registers. *arXiv:2309.16588*, 2023. 4.2, 4.6, 4.6, 4.7, 4.9

[192] Maxime Oquab, Timothée Darcet, Théo Moutakanni, et al. DINOv2: Learning Robust Visual Features without Supervision. *arXiv:2304.07193*, 2024. 4.6

[193] Kaiming He, Xinlei Chen, Saining Xie, et al. Masked Autoencoders Are Scalable Vision Learners. *arXiv:2111.06377*, 2021. 4.6, 4.8.10

[194] Yelysei Bondarenko, Markus Nagel, Tijmen Blankevoort. Quantizable Transformers: Re-

moving Outliers by Helping Attention Heads Do Nothing. *arXiv preprint arXiv:2306.12929*, 2023. 4.7

[195] Kevin Clark, Urvashi Khandelwal, Omer Levy, et al. What Does BERT Look At? An Analysis of BERT's Attention. *arXiv preprint arXiv:1906.04341*, 2019. 4.7

[196] Olga Kovaleva, Alexey Romanov, Anna Rogers, et al. Revealing the Dark Secrets of BERT. *arXiv preprint arXiv:1908.08593*, 2019. 4.7

[197] Alec Radford, Jong Wook Kim, Chris Hallacy, et al. Learning Transferable Visual Models From Natural Language Supervision. *arXiv preprint arXiv:2103.00020*, 2021. 4.6

[198] Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, et al. Gemma 2: Improving Open Language Models at a Practical Size. *arXiv preprint arXiv:2408.00118*, 2024. 2

[199] OpenAI. Learning to Reason with Large Language Models. `https://openai.com/index/learning-to-reason-with-llms/`, 2024. [cite: 227]

[200] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, et al. Gemini: A Family of Highly Capable Multimodal Models. *CoRR*, 2023.

[201] Tom Gunter, Zirui Wang, Ruoming Pang, et al. Apple Intelligence Foundation Language Models. *CoRR*, 2024.

[202] Karl Cobbe, Vineet Jain Kosaraju, Mohammad Bavarian, et al. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021.

[203] Rishita Anubhai Patel, Pushpak Bhattacharyya. Are NLP Models really able to Solve Simple Math Word Problems? *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics*, 2021.

[204] Aida Amini, Saadia Gabriel, Peter Lin, et al. MathQA: Towards Interpretable Math Word Problem Solving with Operation-Based Formalisms. *Proceedings of NAACL*, 2019.

[205] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, et al. Efficient Memory Management for Large Language Model Serving with PagedAttention. *arXiv preprint arXiv:2309.06180*, 2023.

[206] Ari Holtzman, Jan Buys, Li Du, et al. The Curious Case of Neural Text Degeneration. *arXiv preprint arXiv:1904.09751*, 2020.

[207] Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, et al. LLM2Vec: Large Language Models Are Secretly Powerful Text Encoders. *COLM*, 2024. 5.3, 5.3.2

[208] Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *Journal of Machine Learning Research*, 2020. 5.3.2, 5.7.3

[209] Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, et al. Sentence-T5: Scalable Sentence Encoders from Pre-trained Text-to-Text Models. *arXiv preprint arXiv:2108.08877*, 2021.

[210] Hadas Kotek, Rikker Dockum, David Q. Sun. Gender bias and stereotypes in Large Language Models. *arXiv preprint arXiv:2308.14921*, 2023.

[211] Common Crawl. Common Crawl. `https://commoncrawl.org/overview`.

[212] Minyoung Huh, Brian Cheung, Tongzhou Wang, et al. The Platonic Representation Hypothesis. *ICML*, 2024.

[213] Erving Goffman. The presentation of self in everyday life. *Social theory re-wired*, 1959.

[214] Charles Taylor. Sources of the Self: The Making of the Modern Identity. *Harvard University Press*, 1989.

[215] Carl R. Rogers. On Becoming a Person. *Houghton Mifflin*, 1961.

[216] Kaiming He, Xiangyu Zhang, Shaoqing Ren, et al. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. *ICCV*, 2015.

[217] Philip Resnik. Large Language Models are Biased Because They Are Large Language Models. *arXiv preprint arXiv:2406.13138*, 2024.

[218] Tiancheng Hu, Yara Kyrychenko, Steve Rathje, et al. Generative language models exhibit social identity biases. *Nature Computational Science*, 2024.

[219] Alberto Acerbi, Joseph M. Stubbersfield. Large language models show human-like content biases in transmission chain experiments. *Proceedings of the National Academy of Sciences*, 2023.

[220] Emilio Ferrara. Should ChatGPT be Biased? Challenges and Risks of Bias in Large Language Models. *arXiv preprint arXiv:2304.03738*, 2023.

[221] Isabel O. Gallegos, Ryan A. Rossi, Joe Barrow, et al. Bias and Fairness in Large Language Models: A Survey. *arXiv preprint arXiv:2309.00770*, 2023.

[222] Ruby Ostrow, Adam Lopez. LLMs Reproduce Stereotypes of Sexual and Gender Minorities. *arXiv preprint arXiv:2501.05926*, 2025.

[223] Guilherme Penedo, Hynek Kydlíček, Loubna Ben Allal, et al. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. *arXiv preprint arXiv:2406.17575*, 2024. 5.3.1

[224] Ruiqi Zhong, Heng Wang, Dan Klein, et al. Explaining Datasets in Words: Statistical Models with Natural Language Parameters. *NeurIPS*, 2024. 5.6

[225] Ning Ding, Yulin Chen, Bokai Xu, et al. Enhancing Chat Language Models by Scaling High-quality Instructional Conversations. *EMNLP*, 2023. 5.3.1

[226] Yafu Li, Qintong Li, Leyang Cui, et al. MAGE: Machine-generated Text Detection in the Wild. *arXiv preprint arXiv:2305.13242*, 2023.

[227] Rowan Zellers, Ari Holtzman, Hannah Rashkin, et al. Defending Against Neural Fake News. *arXiv preprint arXiv:1905.12616*, 2019.

[228] Eric Mitchell, Yoonho Lee, Alexander Khazatsky, et al. DetectGPT: Zero-Shot Machine-Generated Text Detection using Probability Curvature. *ICML*, 2023. 5.2, 5.6

[229] Hope McGovern, Rickard Stureborg, Yoshi Suhara, et al. Your Large Language Models Are Leaving Fingerprints. *ICCL Workshops*, 2024. 5.6

[230] Sameer Badaskar, Sachin Agarwal, Shilpa Arora. Identifying Real or Fake Articles: Towards better Language Modeling. *IJCNLP*, 2008. 5.6

[231] Lisa Dunlap, Yuhui Zhang, Xiaohan Wang, et al. Describing differences in image sets with natural language. *CVPR*, 2024. 5.6

[232] Zebin You, Xinyu Zhang, Hanzhong Guo, et al. Are Images Indistinguishable to Humans Also Indistinguishable to Classifiers? *arXiv preprint arXiv:2405.18029*, 2024. 5.6

[233] Ruiqi Zhong, Peter Zhang, Steve Li, et al. Goal Driven Discovery of Distributional Differences via Language Descriptions. *arXiv preprint arXiv:2302.14233*, 2023.

[234] Irena Gao, Percy Liang, Carlos Guestrin. Model Equality Testing: Which Model Is This API Serving? *ICLR*, 2025. 5.6

[235] Lisa Dunlap, Krishna Mandal, Trevor Darrell, et al. VibeCheck: Discover and Quantify Qualitative Differences in Large Language Models. *ICLR*, 2025. 5.6

[236] Boya Zeng, Yida Yin, Zhuang Liu. Understanding Bias in Large-Scale Visual Datasets. *NeurIPS*, 2024. 5.2, 5.6

[237] Youssef Mansour, Reinhard Heckel. Measuring Bias of Web-filtered Text Datasets and Bias Propagation Through Training. *arXiv preprint arXiv:2412.02857*, 2024. 5.6

[238] Antonio Torralba, Alexei A. Efros. Unbiased look at dataset bias. *CVPR*, 2011. 5.6

[239] OpenAI. New AI Classifier for Indicating AI-Written Text. `https://openai.com/index/new-ai-classifier-for-indicating-ai-written-text/`, 2024.

[240] OpenAI. GPT-2 Output Dataset Detection. `https://github.com/openai/gpt-2-output-dataset`, 2019. 5.6

[241] Sebastian Gehrmann, Hendrik Strobel, Alexander M. Rush. GLTR: Statistical Detection and Visualization of Generated Text. *ACL*, 2019. 5.6

[242] Marah Abdin, Jyoti Aneja, Harkirat Behl, et al. Phi-4 Technical Report. *arXiv preprint arXiv:2412.08905*, 2024. 5.5, 5.5

[243] Kalpesh Krishna, Yixiao Song, Marzena Karpinska, et al. Paraphrasing evades detectors of AI-generated text, but retrieval is an effective defense. *NeurIPS*, 2023. 5.2

[244] Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, et al. Can AI-Generated Text be Reliably Detected? *TMLR*, 2025. 5.2

[245] Ganesh Jawahar, Muhammad Abdul-Mageed, Laks V.S. Lakshmanan. Automatic Detection of Machine Generated Text: A Critical Survey. *ICCL*, 2020. 5.2

[246] Thomas Lavergne, Tanguy Urvoy, François Yvon. Detecting fake content with relative entropy scoring. *International Conference on Uncovering Plagiarism, Authorship and Social Software Misuse*, 2008. 5.6

[247] Guangsheng Bao, Yanbin Zhao, Zhiyang Teng, et al. Fast-DetectGPT: Efficient Zero-Shot Detection of Machine-Generated Text via Conditional Probability Curvature. *ICLR*, 2024. 5.6

[248] Adaku Uchendu, Zeyu Ma, Thai Le, et al. TURINGBENCH: A Benchmark Environment for Turing Test in the Age of Neural Text Generation. *EMNLP*, 2021. 5.6

[249] Daphne Ippolito, Daniel Duckworth, Chris Callison-Burch, et al. Automatic Detection of

Generated Text is Easiest when Humans are Fooled. *ACL*, 2020. 5.6

[250] John Houvardas, Efstathios Stamatatos. N-gram feature selection for authorship identification. *International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, 2006.

[251] Chi Sun, Xipeng Qiu, Yige Xu, Xuanjing Huang. How to Fine-Tune BERT for Text Classification? *arXiv preprint arXiv:1905.05583*, 2020. 5.3

[252] Efstathios Stamatatos. A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 2009.

[253] Adaku Uchendu, Thai Le, Kai Shu, et al. Authorship Attribution for Neural Text Generation. *EMNLP*, 2020. 5.6

[254] Baixiang Huang, Canyu Chen, Kai Shu. Authorship Attribution in the Era of LLMs: Problems, Methodologies, and Challenges. *arXiv preprint arXiv:2408.08946*, 2024. 5.6

[255] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, et al. Textbooks Are All You Need. *arXiv preprint arXiv:2306.11644*, 2023.

[256] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, et al. Textbooks Are All You Need II: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

[257] Microsoft Research. Phi-2: The Surprising Power of Small Language Models. `https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/`, 2024.

[258] Ruibo Liu, Jerry Wei, Fangyu Liu, et al. Best Practices and Lessons Learned on Synthetic Data. *COLM*, 2024. 5.5

[259] Jerry Wei, Chengrun Yang, Xinying Song, et al. Long-form factuality in large language models. *arXiv preprint arXiv:2403.18802*, 2024.

[260] Ethan Perez, Sam Ringer, Kamilė Lukošiūtė, et al. Discovering Language Model Behaviors with Model-Written Evaluations. *arXiv preprint arXiv:2212.09251*, 2022.

[261] Qwen :, An Yang, Baosong Yang, et al. Qwen2.5 Technical Report. *arXiv preprint arXiv:2412.15115*, 2024. 5.2, 2

[262] Ganqu Cui, Lifan Yuan, Ning Ding, et al. UltraFeedback: Boosting Language Models with Scaled AI Feedback. *arXiv preprint arXiv:2310.01377*, 2023.

[263] Google. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024. 1

[264] Colin White, Samuel Dooley, Manley Roberts, et al. LiveBench: A Challenging, Contamination-Free LLM Benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

[265] Zhuang Liu, Kaiming He. A Decade's Battle on Dataset Bias: Are We There Yet? *ICLR*, 2025. 5.2, 5.6

[266] Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. *ACL*, 2004. 5.4

[267] Tianyi Zhang, Varsha Kishore, Felix Wu, et al. BERTScore: Evaluating Text Generation with BERT. *ICLR*, 2020. (document), 5.7

[268] Tianqi Chen, Carlos Guestrin. Xgboost: A scalable tree boosting system. *KDD*, 2016.

[269] Matthew E. Peters, Mark Neumann, Mohit Iyyer, et al. Deep contextualized word representations. *NAACL*, 2018. 5.3.2

[270] Sepp Hochreiter, Jürgen Schmidhuber. Long Short-term Memory. *Neural Computation MIT-Press*, 1997.

[271] Yu Meng, Mengzhou Xia, Danqi Chen. SimPO: Simple Preference Optimization with a Reference-Free Reward. *NeurIPS*, 2024.

[272] Lewis Tunstall, Edward Beeching, Nathan Lambert, et al. Zephyr: Direct Distillation of LM Alignment. *COLM*, 2024.

[273] Marah Abdin, Jyoti Aneja, Hany Awadalla, et al. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *arXiv preprint arXiv:2404.14219*, 2024. 5.5

[274] Loubna Ben Allal, Anton Lozhkov, Guilherme Penedo, et al. Cosmopedia. `https://huggingface.co/datasets/HuggingFaceTB/cosmopedia`, 2024. 5.3.1

[275] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, et al. LMSYS-Chat-1M: A Large-Scale Real-World LLM Conversation Dataset. *ICLR*, 2024. 5.3.1

[276] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, et al. Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference. *arXiv preprint arXiv:2403.04132*, 2024. 5.5

[277] Yangsibo Huang, Milad Nasr, Anastasios Angelopoulos, et al. Exploring and Mitigating Adversarial Manipulation of Voting-Based Leaderboards. *arXiv preprint arXiv:2501.07493*, 2025. 5.5

[278] Wenting Zhao, Xiang Ren, Jack Hessel, et al. WildChat: 1M ChatGPT Interaction Logs in the Wild. *ICLR*, 2024. 5.3.1