ABSTRACT

Title of Dissertation: UNDERSTANDING AND EXPLOITING WHAT NEURAL NETWORKS LEARN: UNLEARNABLE DATASETS AND INACTIVE ATTENTION HEADS

Pedro Sandoval Segura
Doctor of Philosophy, 2026

Dissertation Directed by: Professor Tom Goldstein
Department of Computer Science

Professor David Jacobs
Department of Computer Science

Neural networks have become the dominant technique across domains due to their ability to find statistical patterns that generalize beyond their training data. However, they also exhibit learning biases and learned behaviors that can be understood and exploited. This dissertation investigates two complementary questions: what patterns neural networks preferentially learn during training, and what computations they perform unnecessarily during inference. We study the first question in the image domain, using primarily convolutional neural networks. We study the second question in the text domain, using pretrained transformer language models. We develop methods that both manipulate learning behavior and identify new insights into transformer computation.

The first part of this dissertation focuses on dataset security through unlearnable datasets.

We study why certain perturbations prevent generalization, showing that perturbations which are learned quickly by convolutional neural networks are most effective at causing overfitting. Building on this insight, we introduce autoregressive perturbations, a model-free and backpropagation-free method for generating reusable unlearnable datasets that remain effective against common defenses. We then examine the limitations of perturbation-based data protection, demonstrating that neural networks can still learn useful representations from ostensibly unlearnable data. We develop methods based on orthogonal projection and image sharpening that recover performance on both bounded and highly corrupted unlearnable datasets, and further show that JPEG compression similarly defeats protections against AI image editing. Together, these results challenge the long-term viability of imperceptible perturbation-based approaches to dataset security.

The second part of this dissertation investigates inefficiencies in transformer language models. We study the attention sink phenomenon and develop score functions for identifying inactive attention heads. By ablating attention heads, our analysis reveals that substantially more attention heads are inactive than previously estimated, allowing more than 12% of attention heads to be zeroed during inference while maintaining model performance. These findings suggest opportunities for reducing transformer inference costs and motivate future architectures that dynamically allocate attention computation. Collectively, we demonstrate that understanding inductive biases and computational structure learned by neural networks provides a foundation for both attacking and improving modern neural networks.

UNDERSTANDING AND EXPLOITING WHAT NEURAL NETWORKS LEARN: UNLEARNABLE DATASETS AND INACTIVE ATTENTION HEADS

by

Pedro Sandoval Segura

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
August 3, 2026

Advisory Committee:
Professor Tom Goldstein, *Chair & Advisor*
Professor David Jacobs, *Co-Advisor*
Professor William Regli
Professor Ming Lin
Professor Behtash Babadi

© Copyright by
Pedro Sandoval Segura
2026

Acknowledgments

I have the privilege of writing this dissertation because of the sacrifices made by my parents, Martin Sandoval and Otilia Segura. Their focus on education, working hard, and trying my best no matter the circumstances, has led me to where I am today. I'm lucky to have you both as my parents, and equally lucky to have my sister, Marisol, whose advice, care, and friendship made me a better person.

The content of this dissertation would not have been possible without the advice and guidance of my advisors, Prof. David Jacobs and Prof. Tom Goldstein. In 2021, as I switched research areas, Prof. Jacobs invited me to his group meetings and helped me build a foundation in computer vision and deep learning. I am grateful for your patience as I made mistakes, learned the basics, and developed my own research interests. Looking through old emails, I am surprised and deeply grateful for the care with which you read my preprints, slides, and other writing, no matter how small the occasion. In 2021, I also met Prof. Tom Goldstein over Zoom, during the pandemic, to discuss research ideas. At the time, I remember thinking, "Why would a professor share so many of his promising ideas with a random student?" It's clear to me now that you enjoy thinking about new problems with *all students*, to get them to find a research direction they love. Working with Tom Lab has been one of the most enjoyable parts of my PhD, and I will miss thinking of "crazy ideas" with everyone. I learned so much, academically and personally, from many members of the lab: Vasu Singla, Jonas Geiping, Micah Goldblum, Liam Fowl, and Ashwinee Panda. I am fortunate to have had the opportunity to work alongside each of you. I also want to thank Prof. Bill Regli, who consistently checked in on me and encouraged me to

pursue opportunities and fellowships that I never thought I could earn, but ultimately did.

I likely would not have considered doing research if it weren't for my college roommate and friend, Mehdi Drissi. Your interest in deep learning was contagious and I'm thankful you were willing to include me in your research group. I am also thankful for the guidance and encouragement of great professors from Harvey Mudd College: Prof. Ben Wiedermann, Prof. Robert Keller, and Prof. George Montañez. Each of you showed me how exciting scientific research could be and inspired me to push forward.

Whenever I traveled back to San Francisco, I had the constant support and encouragement of my high school benefactor, John D. Moriarty. For more than 11 years, he has taken a genuine interest in my education. He loves learning about my classes, research, career goals, and predictions for the field of AI. He is like a father figure to me, standing by me during difficult times, and celebrating with me during my successes. He has never stopped believing in my potential. I hope that, in my career, I can have even a fraction of the positive impact that he has had on so many students.

Whenever I left College Park, I missed being back in College Park because of my roommates. You guys made coming home something to look forward to: Alejandro, Ruben, Juan, Suraj, Alvaro, Xijun, Nati, and Kasun. I'm going to miss our movie nights, dinners, and late night talks.

Finally, I'd like to thank my fiancée, Emily. Throughout this degree, you've watched me be happy, excited, frustrated, sad, and everything in between. You helped me find solutions where I thought there weren't any. You made life exciting, finding new places to eat, new places to see, new activities to do. From the beginning of my time in Maryland, when you helped me set up my room, to the very end, when you organized my job search, you are always there looking for ways to make my life easier. Thank you for making this research possible.

Table of Contents

Acknowledgements ii

Table of Contents iv

List of Tables viii

List of Figures x

List of Abbreviations xii

Chapter 1: Introduction 1
- 1.1 Outline . . . 3
  - 1.1.1 Part I: Making Unlearnable Datasets using Learning Biases . . . 3
  - 1.1.2 Part II: The Limits of Perturbation-based Protection . . . 4
  - 1.1.3 Part III: Unnecessary Computation During Inference . . . 6
- 1.2 Summary of Contributions . . . 7
  - 1.2.1 Part I . . . 7
  - 1.2.2 Part II . . . 8
  - 1.2.3 Part III . . . 10
- 1.3 Chapter Map . . . 11

Chapter 2: Literature Review 12
- 2.1 Dataset Poisoning . . . 12
- 2.2 Unlearnable Datasets . . . 14
  - 2.2.1 Optimized Perturbations . . . 16
  - 2.2.2 Synthetic Perturbations . . . 17
  - 2.2.3 Attacks . . . 18
  - 2.2.4 Analysis . . . 19
- 2.3 Learning Shortcuts and Simplicity Bias . . . 20
- 2.4 Robustness . . . 21
- 2.5 Transformers . . . 22
  - 2.5.1 Interpretability . . . 23
  - 2.5.2 Attention Sinks . . . 23

I Making Unlearnable Datasets using Learning Biases **25**

Chapter 3: Poisons That are Learned Faster are More Effective 26
3.1 Experimental Setup . . . 27
3.1.1 Optimized Poisons . . . 27
3.1.2 Synthetic Poisons . . . 29
3.1.3 Training Hyperparamters . . . 31
3.2 Learning Speed Determines Peak Test Accuracy . . . 31
3.3 Stronger Adversarial Attacks Do Not Make Stronger Poisons . . . 34
3.4 Conclusion . . . 35

Chapter 4: Autoregressive Perturbations for Data Poisoning 37
4.1 Why Autoregressive Perturbations? . . . 38
4.2 Generating Autoregressive Perturbations . . . 40
4.3 A Two-layer, Manually-specified CNN Can Classify AR Perturbations . . . 44
4.4 Finding AR Process Coefficients . . . 46
4.5 Experimental Setup . . . 49
4.5.1 Datasets and Models . . . 50
4.5.2 Training Hyperparameters . . . 51
4.6 AR Perturbations are Dataset and Architecture Independent . . . 51
4.7 AR Perturbations Work Against Common Defenses . . . 53
4.7.1 Against Data Augmentations and Smaller Perturbations . . . 53
4.7.2 Against Adversarial Training . . . 55
4.7.3 Against Mixing with Clean Data . . . 56
4.8 AR Poisoning in ℓ_∞ . . . 57
4.9 Effect of Optimizer Hyperparameters . . . 58
4.10 Conclusion . . . 59
4.11 Appendix . . . 60
4.11.1 Samples of CIFAR-10 Poisons . . . 60
4.11.2 Dataset Details . . . 60

II The Limits of Perturbation-based Protection **64**

Chapter 5: What Can We Learn From Unlearnable Datasets? 65
5.1 Experimental Setup . . . 66
5.1.1 Datasets and Models . . . 66
5.1.2 Training Hyperparameters . . . 68
5.2 Learning Useful Features from Unlearnable Datasets . . . 69
5.2.1 Additional Model Architectures . . . 72
5.2.2 Additional Base Datasets . . . 73
5.3 Linearly Separable Perturbations are Not Necessary . . . 74
5.3.1 Evaluating Linear Separability of Poison Images . . . 77
5.4 Learning from Unlearnable Datasets with Class-wise Perturbations . . . 77
5.4.1 Orthogonal Projection Method . . . 78

5.4.2 Orthogonal Projection Results . . . 80
5.4.3 Visualizing Logistic Regression Weights . . . 84
5.4.4 Samples from Unlearnable Datasets . . . 85
5.5 Conclusion . . . 85

Chapter 6: Learning From Convolution-based Unlearnable Datasets 92
6.1 Introduction to Unbounded Unlearnable Datasets . . . 93
6.2 Related Work . . . 94
6.3 Motivation . . . 97
6.3.1 Shortcut Learning . . . 97
6.3.2 Frequency Filtering via Discrete Cosine Transform . . . 99
6.4 Method and Experiments . . . 100
6.4.1 Settings . . . 100
6.4.2 Random Sharpening Kernels + Frequency Filtering . . . 101
6.4.3 Analysis on Frequency Filtering . . . 103
6.5 Conclusion . . . 104

Chapter 7: JPEG Compressed Images Can Bypass Protections Against AI Editing 106
7.1 Protection Methods . . . 106
7.1.1 Photoguard: Protecting Against Malicious AI-Powered Image Editing . . . 107
7.1.2 Glaze: Protecting Against Style Mimicry . . . 110
7.2 Bypassing Photoguard Protection . . . 110
7.2.1 JPEG Compression as a Low-Cost Transform . . . 111
7.2.2 Gaussian and Median Blurs of the Protected Image . . . 114
7.2.3 Rotations and Horizontal Flip of the Protected Image . . . 114
7.3 Conclusion . . . 115

III Unnecessary Computation During Inference 118

Chapter 8: Identifying and Evaluating Inactive Heads in Pretrained LLMs 119
8.1 Introduction . . . 120
8.2 Background and Related Work . . . 122
8.3 Evaluating Score Functions for Inactive Heads . . . 125
8.4 Experiments . . . 129
8.4.1 Setup . . . 129
8.4.2 Score Functions Identify Different Sets of Heads . . . 130
8.4.3 Verifying Inactive Heads through Model Interventions . . . 132
8.4.4 Inactive Heads by Layer . . . 135
8.4.5 Score Distributions for Studying Attention Behaviors . . . 136
8.4.6 Circuit Definition of Attention Head Output . . . 138
8.4.7 Sample Inactive Heads Identified by Score Functions . . . 140
8.5 Conclusion . . . 141
8.6 Appendix . . . 142
8.6.1 Full Results of Sec. 8.4.3 . . . 142

8.6.2 Multiple Forward Pass Analysis . . . 144
8.6.3 Score Distributions on MMLU . . . 146
8.6.4 Full Results of Sec. 8.4.5 Wasserstein Distances . . . 146
8.6.5 Score Distributions Comparison: FineWeb-Edu vs. MMLU . . . 146
8.6.6 Attention sinks and value-state drains . . . 146
8.6.7 Redundancy of attention patterns . . . 146
8.6.8 On lm-evaluation-harness evaluations . . . 148
8.6.9 PyTorch Implementation of Avg Head Output Norm (LN) and Avg Weight to First Token . . . 148
8.6.10 Additional model and dataset information . . . 150
8.6.11 Practical Implications and Limitations . . . 151

Chapter 9: Conclusion 163
9.1 Future Work . . . 165
9.1.1 Sharing Data Using Latent Codes . . . 165
9.1.2 Transformer Efficiency . . . 166

Bibliography 168

List of Tables

1.1 Chapter Map . . . 11

3.1 A summary of all CIFAR-10 poisons considered in initial work . . . 30
3.2 Attack transferability of different error-maximizing CIFAR-10 poisons against victim models of varying architecture . . . 33

4.1 Dataset Independence of Autoregressive Poisoning . . . 52
4.2 Architecture Independence of Autoregressive Poisoning . . . 53
4.3 Autoregressive Poisoning against Strong Data Augmentations . . . 54
4.4 Autoregressive Poisoning against Adversarial Training . . . 56
4.5 Autoregressive Poisoning against Mixing . . . 57
4.6 Autoregressive Poisoning in ℓ_∞ . . . 58
4.7 Effect of Optimizer and LR on AR Poisoning . . . 58

5.1 Generalizable features can be learned from unlearnable datasets . . . 69
5.2 Generalizable features can be learned using a variety of network architectures. . . . 73
5.3 Generalizable features can be learned from different underlying distributions . . . 74
5.4 Linearly separable perturbations are not necessary to create unlearnable datasets . 75
5.5 Linear separability of poisoned images . . . 77
5.6 Subtracting the average class image is not effective . . . 80
5.7 Orthogonal Projection can make class-wise unlearnable data learnable . . . 81
5.8 Orthogonal Projection can make class-wise unlearnable data learnable for ViT . . 83
5.9 Orthogonal Projection is competitive on CIFAR-100 . . . 84

6.1 Random Sharpening Kernels and Frequency Filtering (RSK+FF) applied to CUDA data . . . 93
6.2 Size of an average perturbation from different unlearnable datasets . . . 96
6.3 Unlearnable datasets train models to learn perturbations . . . 98
6.4 Breakdown of the transforms we use against CUDA . . . 102
6.5 RSK is better than SSK . . . 103
6.6 Random sharpening kernels on CUDA CIFAR-100 . . . 104
6.7 Random sharpening kernels on CUDA ImageNet-100 . . . 104
6.8 Random sharpening kernels on CUDA CIFAR-10 . . . 105

8.1 Equations of Inactive Head Score Functions . . . 126

8.2 More than 12% of attention heads can be zeroed while keeping average accuracy within 1% of the original model 133
8.3 Circuit-based definition of head output 140
8.4 Inactive Heads on PIQA Benchmark 143
8.5 Inactive Heads on WinoGrande Benchmark 144
8.6 Additional model information of pretrained LLMs used in this work. 150

List of Figures

3.1 Poisons which are learned faster have lower peak test accuracy on CIFAR-10 . . 27
3.2 Final test accuracy doesn't correlate well with peak test accuracy 28
3.3 Normalized perturbations from adversarial and synthetic poisons 29
3.4 Train loss and test accuracy curves when training a ResNet-18 on a variety of poisons 31

4.1 Overview of Autoregressive Poisoning 38
4.2 Generating Autoregressive Perturbations 41
4.3 Comparison of poisoned images from various poisons 43
4.4 A closeup of a CIFAR-10 image from our AR poison 43
4.5 A two-layer CNN can perfectly classify AR perturbations 44
4.6 Samples from Error-Max Poison 61
4.7 Samples from Error-Min Poison 61
4.8 Samples from Regions-4 Poison 62
4.9 Samples from Regions-16 Poison 62
4.10 Samples from our AR Poison 63

5.1 Overview of challenges for unlearnable datasets 66
5.2 Features learned during poison training can be reweighted for high test accuracy . 87
5.3 Logistic regression learns linearly separable features 88
5.4 Average class images vaguely display class-wise perturbations 88
5.5 Logistic regression weights from sample-wise unlearnable datasets 89
5.6 Logistic regression weights from class-wise unlearnable datasets 89
5.7 Samples from sample-wise CIFAR-10 unlearnable datasets 90
5.8 Samples from class-wise CIFAR-10 unlearnable datasets 90
5.9 Samples from SVHN unlearnable datasets 91
5.10 Samples from CIFAR-100 unlearnable datasets 91

6.1 Sharpening and Frequency Filtering of a CIFAR-10 Image 94
6.2 Comparison of the same image from different unlearnable datasets 97
6.3 DCT can be used to remove exact frequency bands 99
6.4 Building a random sharpening kernel 101
6.5 Image reconstructions using different percentages of frequencies kept 104

7.1 JPEG compression allows an adversary to modify a protected image found online 109

7.2 More JPEG compression more reliably undermines photoguard protection 111
7.3 JPEG compression allows an adversary to edit the background of a protected image found online 112
7.4 More JPEG compression more reliably undermines photoguard protection 113
7.5 Gaussian blurs of photoguard images 115
7.6 Median blurs of photoguard images 116
7.7 Rotations of photoguard images 117
7.8 Flip of photoguard image 117

8.1 Score Functions for Inactive Heads 121
8.2 Score functions identify different sets of heads 123
8.3 Measuring Avg Head Output Norm is best at identifying inactive heads for most models 131
8.4 Inactive heads can be identified and zeroed with minor performance degradation . 132
8.5 Average Head Output Norm is a stable measure layer-wise inactivity 135
8.6 Head score distributions can be similar within a model family 137
8.7 Circuit-based definition of head output 140
8.8 Sample Attention Heads for 6 Unnormalized Score Functions 153
8.9 Sample Attention Heads using Avg Head Output Norm (LN) 154
8.10 Complete MMLU Results of Sec. 8.4.3 155
8.11 Additional PIQA Results for Sec. 8.4.3 156
8.12 Additional WinoGrande Results for Sec. 8.4.3 157
8.13 Inactive Heads as Sequence Length Changes 158
8.14 Inactive Heads as Sequence Length Changes 158
8.15 Inactive Heads as Sequence Length Changes 158
8.16 Head Score Distributions on MMLU 159
8.17 Wasserstein Distance of Head Score Distributions between Models 160
8.18 Distribution of Avg Weight of First Token Scores changes when the input distribution changes 161
8.19 Attention sink tokens have the smallest value vector norms 161
8.20 PCA of Attention Matrices 162

List of Abbreviations

DNN	Deep Neural Network
CNN	Convolutional Neural Network
FC	Fully-connected Layer
SGD	Stochastic Gradient Descent
RN-18	ResNet-18
ViT	Vision Transformer
VAE	Variational Autoencoder
VQ-VAE	Vector Quantized-Variational Autoencoder
LLM	Large Language Model
MLP	Multi-layer Perceptron
CV	Computer Vision
NLP	Natural Language Processing
FGSM	Fast Gradient Sign Method
MI-FSGM	Momentum Iterative Fast Gradient Sign Method
PGD	Projected Gradient Descent
AR(p)	Autoregressive process of degree p
DCT	Discrete Cosine Transform
RSK	Random Sharpening Kernel
NaN	Not a Number
NTP	Next Token Prediction

Chapter 1: Introduction

At the Crater of Diamonds State Park in Murfreesboro, Arkansas, any rock or mineral you find is yours to keep, so it is no surprise that rain or shine, tourists and locals alike pan across the vast 37-acre field hoping for their chance at the rare mineral. But being inexperienced, and possibly under the hope of striking it rich, you assume anything translucent and shiny could be a diamond. Rather than reasoning about a rock's hardness, this *shortcut*, or simple decision rule, leads you to save all the shiny rocks you find, which you collect in a small paper bag. Fortunately, on the way out, park staff provide complimentary identification of all the quartz, agate, and jasper you found.

Like tourists classifying rocks in Arkansas, pattern recognition systems also make mistakes due to simple decision rules [1]. For example, a convolutional neural network (CNN) trained to perform image recognition may appear to perfectly classify images of cows, but fail on images where cows appear on the beach, outside of the typical grassy landscape, revealing the unintended shortcut of "grass" as a predictor for "cow" [2]. In classifying pneumonia from X-ray scans, CNNs may superficially achieve good performance, but struggle when the scans come from a different hospital. Rather than focusing on the lungs subregion of the scan, CNNs can unintentionally learn to detect a hospital-specific metal token common in scans from a hospital with high pneumonia prevalence [3].

Even transformer language models suffer from simple decision rules. For example, when tasked with reading comprehension questions, transformers often rely on lexical overlap: predicting an answer simply because it shares words with the prompt or failing when irrelevant sentences are introduced [4]. Similarly, language models can be easily manipulated by misleading prompts into confidently stating false information and misconceptions, demonstrating a dependence on frequent patterns in training data rather than robust reasoning [5, 6]. Language models have also been shown to use superficial shortcuts, like assuming any sentence containing the word "not" implies a negative relationship, and many other heuristics [7].

Due, in part, to the training objective, deep neural networks (DNNs) can leverage unintended patterns in data if they are correlated with ground truth labels. This learning preference for simple patterns in data can be a drawback in practice, but can be relied upon if one's objective is to prevent DNNs from learning representations that generalize. For instance, the prevalence of data scraping from social media as a means to obtain datasets has led to growing concerns regarding unauthorized use of data. By adding small, imperceptible perturbations to data, a data owner can make data "unlearnable" or useless for training DNNs despite the data being seemingly unchanged. If data is filled with learning shortcuts, it would be more prudent for a web scraper to refrain from using it, thereby enabling the data owner to restrict the use of their data.

Learning shortcuts arise across data domains, not from a lack of model capacity, but from a surplus of an easy signal: when a simple input feature correlates with a target output during training, gradient descent has no reason to look further. Although learning shortcuts prevent generalization, achieving generalization does not eliminate learned behaviors that may themselves be a different form of shortcut. One example is how transformer neural networks learn to perform unnecessary forward pass computations. Together, these observations motivate two

complementary questions we investigate in this dissertation: what patterns do neural networks prefer, and what computations do they perform unneccessarily at inference time? If we better understand preferred patterns, we can craft better "unlearnable" datasets. If we can understand how unnecessary computations work, we can provide insights into reducing inference computation.

1.1 Outline

1.1.1 Part I: Making Unlearnable Datasets using Learning Biases

Part I takes up the first question of this dissertation: What patterns do neural networks preferentially learn from data? Can that preference be controlled? Dataset security is the main application for learning biases or "shortcuts" that we probe and improve in this work. By rendering data useless, a data owner can decide whether to allow or restrict its use, essentially choosing to secure the data or not. For creating unlearnable data, we examine the types of perturbations that are effective, the reasons behind their effectiveness, and the circumstances in which they may fail.

In the context of image classification, Chapter 3 studies a number of additive patterns and perturbations on natural images. Because the perturbations are predictive but absent from the test set, understanding what patterns neural networks prefer is a matter of measuring degradation in test accuracy. This degradation in test accuracy motivates referring to these modified datasets as "poisons" or "unlearnable datasets". Across 25 error-minimizing, error-maximizing, and synthetic perturbations, the variable that predicts a perturbation's effectiveness is neither its norm nor the strength of the attack that produced it, but the number of epochs a network needs to reach a threshold training loss. This also changes how such datasets must be evaluated because a

quickly learned shortcut suppresses accuracy throughout training while a slowly-learned one does not. Peak test accuracy, rather than final test accuracy, is the quantity that protective perturbation must control.

Rather than search for a perturbation that happens to be learned quickly, in Chapter 4 we derive one that must be. Every autoregressive process admits a convolutional filter that produces a zero response. We posit that autoregressive perturbations must be easy to learn partly because, without any training, we can manually specify all the parameters of a two-layer CNN that separates autoregressive signals perfectly. Because our construction appeals to the structure of the convolution operator and not to the contents of any dataset, we show that the same set of autoregressive processes can generate perturbations for new datasets, new image resolutions, and new architectures without re-optimization. Compared to existing unlearnable methods, our autoregressive (AR) poisons are more resistant against common defenses such as adversarial training and strong data augmentations. Relying on the learning bias of CNNs, AR perturbations provide insight into what makes an effective data poison.

Together these chapters give an answer to our initial question: a neural network's learning preference is predictable (Chapter 3), stable, and derivable from structure (Chapter 4). Part II examines how to exploit it.

1.1.2 Part II: The Limits of Perturbation-based Protection

Part II asks the same question as Part I in the opposite direction: If we can characterize what makes a signal easy for a network to learn, can we also characterize what makes it easy to remove?

Chapter 5 begins to answer that question by breaking recent unlearnable dataset methods, where perturbations are additive, bounded, and imperceptible. Low test accuracy, it turns out, does not certify that nothing will be learned: feature extractors trained entirely on unlearnable data support high accuracy once their final layer is retrained on a small clean set, so the standard evaluation does not measure what it is assumed to measure. By finding a counterexample, we refute the prevailing explanation for why unlearnable datasets work: that perturbations are linearly separable. Finally, we show that when perturbations are applied class-wise, they span a subspace of dimension at most the number of classes, which an orthogonal projection removes at a fraction of the cost of adversarial training.

Chapter 6 removes the imperceptibility constraint of perturbations for unlearnable data, which is the natural response to Chapter 5. We present a method for learning from highly-corrupted Convolution-based Unlearnable Datasets (CUDA). CUDA datasets replace additive perturbations with visible class-wise blurs, based on the argument that unbounded perturbations cannot be subtracted away. Nevertheless, we develop sharpening kernels that allow us to learn from CUDA data anyway. Our work here is yet another example contradicting the notion that unlearnable datasets contain no useful information.

Chapter 7 departs from the image classification domain into the image generation domain. Protections against AI image editing defend a single image against a diffusion model rather than a dataset against a classifier, and are optimized end-to-end against the same image generation model they must protect against. Like prior chapters, image protection depends on an imperceptible perturbation. Surprisingly, JPEG compression alone is enough to make a protected image editable again, demonstrating the brittleness of imperceptible perturbations.

The three chapters in Part II differ in what is protected, in how large the perturbation can

be, and in the type of model that operates on these perturbations, but the conclusion does not change: perturbation-based protections are brittle. Having explored what neural networks learn from data, and having exploited those findings to break protective methods, Part III turns to study what a trained networks have learned not to do.

1.1.3 Part III: Unnecessary Computation During Inference

Part III focuses on the second question of this dissertation: What computations do trained networks perform unnecessarily during inference? How do we identify them? The first two parts asked what a network learns from data and whether it could be exploited. This part asks about computations on every input after training is over.

A transformer language model executes every attention head at every layer on every token of every forward pass, but not every head contributes. Some have learned to place nearly all of their attention on the first token of the sequence, a token that encodes the least information. Similar to Part I where we looked for a scalar that predicts which of two signals a network will fit, and found learning speed. Here, we look for the simplest scalar that predicts whether an attention head matters.

Chapter 8 evaluates 12 score functions across 14 pretrained language models from three model families, and verifies them functionally, by ablating the heads they identify. Our score functions depart from the literature, which has primarily analyzed attention weights to determine whether a head is active or inactive. We find that the norm of an attention head's output best measures whether it affects the model's output. More than 12% of heads can be zeroed per forward pass while holding MMLU accuracy within 1% of the pretrained model. The first-token

attention-sink criterion that motivated this line of work identifies fewer than 5%, indicating that we find more unnecessary computation than previously known. Finally, Chapter 9 draws the parts together and proposes suggestions for future work.

Overall, this dissertation provides evidence that understanding how neural networks learn and the behaviors that existing models exhibit is essential. By understanding shortcut learning, we gain the ability to predict and control what a neural network learns. By studying the attention mechanisms in transformer language models, we reveal inactive computation that can be measured and analyzed. Together, these case studies illustrate how explaining learned behavior leads to both practical methods and a deeper understanding of neural networks.

1.2 Summary of Contributions

1.2.1 Part I

Chapter 3: Poisons That are Learned Faster are More Effective

- Using 25 error-minimizing, error-maximizing, and synthetic unlearnable datasets, we show that perturbations reaching a threshold training loss in fewer epochs yield lower *peak* test accuracy. We find learning speed, rather than perturbation norm or attack strength, as the variable that governs effectiveness.

- We show that final test accuracy is a poor proxy for peak test accuracy, and argue that unlearnable datasets must be evaluated by peak test accuracy because early stopping is otherwise a sufficient defense; a state-of-the-art error-maximizing poison is $7\times$ less effective when training is stopped at epoch 8.

- We show that stronger and more transferable adversarial attacks do *not* produce stronger poisons, refuting an assumed tie between evasion-attack strength and poisoning strength.

Chapter 4: Autoregressive Perturbations for Data Poisoning

- We introduce autoregressive (AR) perturbations, generated by AR processes rather than by optimization, requiring no surrogate network, no backpropagation, and no access to the target dataset.
- We show that every AR(p) process admits a convolutional filter that produces a zero response, and we construct by hand a two-layer CNN that classifies AR signals perfectly.
- Because the construction depends on AR structure and not on the data, the same set of K AR processes generates perturbations for any K-class dataset at any image resolution, and transfers across architectures.
- AR poisons are competitive with or stronger than optimized poisons under adversarial training, strong augmentation (Cutout, CutMix, Mixup), architecture change, and mixing with clean data.

1.2.2 Part II

Chapter 5: What Can We Learn From Unlearnable Datasets?

- We show that low test accuracy does not imply that no useful information was acquired: feature extractors trained entirely on unlearnable data support high test accuracy once the final layer is re-fit on a small clean set. Low test accuracy is therefore not a valid audit of data protection.

- We refute the prevailing hypothesis that unlearnable datasets work *because* their perturbations are linearly separable, using AR perturbations as an explicit counterexample. This finding opens the design space for new types of unlearnable data.

- We introduce the Orthogonal Projection attack, which recovers accuracy from class-wise perturbed datasets published at ICML 2021 and ICLR 2023 at a fraction of the cost of adversarial training, establishing that class-wise perturbations cannot be relied upon.

Chapter 6: Learning From Convolution-based Unlearnable Datasets

- We show that abandoning imperceptibility does not rescue protection: random sharpening kernels combined with DCT frequency filtering break the class-wise blurs of the Convolution-based Unlearnable Dataset (CUDA).

- We improve test accuracy over CUDA standard training by 55.3%, 36.0%, and 40.4% on CIFAR-10, CIFAR-100, and ImageNet-100, exceeding adversarial training, and argue that random sharpening should be a required baseline for convolution-based protections.

Chapter 7: JPEG Compressed Images Can Bypass Protections Against AI Editing

- We show that JPEG compression alone allows a diffusion model to successfully edit images "protected" by Photoguard, extending the brittleness result from discriminative training to generative editing.

- We ground our finding in a general argument: if a human can still read an image's semantic content, then a semantics-recovering function demonstrably exists, so imperceptibility and protection are in direct tension for any imperceptible perturbation.

1.2.3 Part III

Chapter 8: Identifying and Evaluating Inactive Heads in Pretrained LLMs

- We formalize 12 score functions covering three distinct notions of inactivity: attention-weight concentration, value-state magnitude, and head-output norm. We threshold these scores to classify attention heads of 14 pretrained LLMs across three model families.
- We validate our score functions functionally, by zeroing identified heads and measuring benchmark accuracy, rather than solely inspecting attention maps.
- We find that score functions measuring a head's average output norm are best: more than 12% of heads can be zeroed per forward pass while keeping MMLU accuracy within 1% of the pretrained model. On PIQA and WinoGrande, our score functions show that more than 21% and 14% of attention heads, respectively, can be zeroed while keeping average accuracy within 1% of the original model.
- We find the standard first-token attention-sink criterion identifies less than 5% of heads, missing more than 7% of heads that are in fact inactive. Thus, we show that attention sinks are a subset of, not a synonym for, inactive computation in transformers.
- We show that score distributions are themselves an analysis tool: they reveal that instruction tuning and preference tuning barely change attention behavior, while model scale changes it only at the largest scales.

1.3 Chapter Map

Ch.	Question it answers	Key result	Data
3	What is learned first?	Learning speed predicts peak test accuracy	CIFAR-10
4	Can we construct it?	AR perturbations that a hand-specified CNN separates	CIFAR, SVHN, STL-10
5	Does protection hold?	Unlearnable data still learnable; Orthogonal Projection	CIFAR, SVHN, ImageNet
6	Does it hold if unbounded?	Sharpening + frequency filtering break CUDA	CIFAR, ImageNet-100
7	Does it hold for generative editing?	JPEG compression breaks Photoguard	Photoguard images
8	What computations are unnecessary?	$> 12\%$ of attention heads are inactive	MMLU, PIQA, WinoGrande

Table 1.1: How each chapter addresses key questions.

Chapter 2: Literature Review

This chapter establishes vocabulary and prior results the rest of the dissertation depends on. Each of the questions posed in Chapter 1 draws on a separate literature. Learning biases are the sole mechanism behind dataset poisoning and unlearnable datasets. We review attacks proposed against them, the analyses offered to explain why they work, and the broader literature on shortcut learning and robustness. For understanding unnecessary computations of transformers, we review transformer interpretability and the attention sink phenomenon. Readers familiar with either literature may skip to Chapter 3 or Chapter 8.

2.1 Dataset Poisoning

Web scraping is increasingly used to construct large datasets, necessary for training groundbreaking deep learning models [8, 9, 10, 11]. But collecting enormous datasets through web scraping makes it intractable for a human to review samples in a meaningful way or to obtain consent from relevant parties [12]. In fact, companies have already trained commercial facial recognition systems using personal data collected from media platforms [13]. To prevent the further exploitation of online data for unauthorized or illegal purposes, imperceptible, adversarial modifications to images can be crafted to cause erroneous output for a neural network trained on the modified data [14]. This crafting of malicious perturbations for the purpose of interfering

with model training is known as dataset poisoning. The modified data, called a *poison*, can induce malicious behavior in a deep neural network trained on this data by an unwitting practitioner [14]. Carlini et al. [15] demonstrate that dataset poisoning is practical and cheap: buying just $\$60$ worth of domains is enough to poison 0.01% of the LAION-400M dataset [16].

One of the earliest instances of a data poisoning attack intended to reduce overall test performance is from Biggio et al. [17], who optimize a single sample that corrupts SVM training. Poisoning a convex model like SVM or logistic regression can be solved exactly, but poisoning deep neural networks is more challenging. A number of approaches have been proposed, validated primarily by empirical evidence due to the complexity of the objective (See Eq. 2.1 and 2.2). Data poisoning attacks are considered a *training-time vulnerability*, as opposed to evasion attacks (*e.g.* adversarial examples) [18, 19, 20, 21, 22] which are a *test-time vulnerability*. Data poisoning attacks are tailored for different kinds of erroneous model behavior: backdoor attacks ensure prespecified input features cause inaccurate output [23, 24], feature-collision (or targeted poisoning) attacks cause a particular target example to be misclassified as a base class [25, 26], and availability attacks aim to degrade overall test performance [17, 27, 28, 29]. Most work in dataset poisoning exploits model behavior when the adversary has control over a small portion of the training dataset, *i.e.* less than 1%. Availability attacks, on the other hand, assume the adversary to be the data owner and have access to the entirety of the training set.

Availability attacks are a flavor of data poisoning attack where the poisoner tries to induce poor performance for the victim network on the clean distribution. This form of data poisoning has been pitched as a means of protecting data privacy and allowing for secure dataset release [30, 31]. A data owner, for example, may want to release a subset of data that they wish not be integrated into a larger dataset. In the context of facial recognition, a number of works

have demonstrated that availability attacks can be used to reduce the accuracy of protected classes [28, 29, 30]. Another perspective on the objective of availability attacks is rendering data useless, implying that a practitioner with access to the data would be better off not using it [32]. Availability attacks [29, 33] are also known as generalization attacks [34], delusive attacks [35], unlearnable datasets [36, 37], or simply unlearnable examples [28, 38].

Terminology Data poisoning methods that perturb the entire training dataset, and which we refer to as "unlearnable datasets" or simply "poisons", are the focus of this work, so we use these terms interchangeably. We also consider unlearnable datasets as defenses, given that their primary use case is to prevent the exploitation of data. Aiming to learn from unlearnable datasets is considered an attack.

2.2 Unlearnable Datasets

In an attempt to prevent the unauthorized use of data, unlearnable dataset methods make small perturbations to training data so that DNNs trained on the modified data result in poor test accuracy [28, 29, 30, 32, 33]. The idea is that if generalization performance is harmed by incorporating the modified, "unlearnable" data, third parties will be disincentivized from scraping it. Unlearnable dataset methods answer the following question: *How should one imperceptibly modify the clean training set to cause the largest generalization gap?*

We consider the problem of creating a clean-label unlearnable dataset in the context of a K-way image classification task, following Huang et al. [28]. We denote the clean training and test datasets as $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{test}}$, respectively. We assume $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{test}}$ are sampled from a common distribution $\sim \mathcal{T}$. The goal is to perturb $\mathcal{D}_{\text{train}}$ into a poisoned dataset $\widetilde{\mathcal{D}}_{\text{train}}$ such that DNNs

trained on the poisoned data perform poorly.

Suppose there are n samples in a clean training set, i.e. $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$ where $x_i \in \mathbb{R}^d$ are the inputs and $y_i \in \{1, ..., K\}$ are the labels. We consider an unlearnable dataset, denoted $\widetilde{\mathcal{D}}_{\text{train}} = \{(x_i', y_i)\}_{i=1}^n$ where $x_i' = x_i + \delta_i$ is the perturbed or poisoned version of the example $x_i \in \mathcal{D}_{\text{train}}$ and where $\delta_i \in \Delta \subset \mathbb{R}^d$ is the perturbation. The set of allowable perturbations, Δ, is typically an ℓ_p-ball with radius ϵ where ϵ is small enough that δ is imperceptible, and does not affect the utility of the example.

Unlearnable datasets are created by applying a perturbation to a clean image in either a *class-wise* or *sample-wise* manner. When a perturbation is applied class-wise, every sample of a given class is perturbed in the same way. That is, $x_i' = x_i + \delta_{y_i}$ and $\delta_{y_i} \in \Delta_C = \{\delta_1, ..., \delta_K\}$. When samples are perturbed in a sample-wise manner, every sample has a unique perturbation. All unlearnable dataset methods aim to solves the following bi-level maximization:

$$\max_{\delta \in \Delta} \mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{test}}} \left[\mathcal{L}(f(x), y; \theta(\delta))\right] \tag{2.1}$$

$$\theta(\delta) = \arg\min_{\theta} \mathbb{E}_{(x_i,y_i)\sim\mathcal{D}_{\text{train}}} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta)\right] \tag{2.2}$$

Equation (2.2) describes the process of training a model on unlearnable data, where θ denotes the model parameters. Equation (2.1) states that the unlearnable data should be chosen so that the trained network has high test loss, and thus fails to generalize to the test set.

2.2.1 Optimized Perturbations

Different approaches have been proposed to construct δ_i. Both error-maximizing [29] and error-minimizing [28] poisoning approaches use a surrogate network, trained on clean training data, to optimize perturbations. We denote surrogate network parameters as θ^*. Error-maximizing poisoning [29] proposes constructing δ_i that maximize the loss of the surrogate network on clean training data:

$$\max_{\delta \in \Delta} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}_{\text{train}}} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta^*) \right] \tag{2.3}$$

whereas error-minimizing poisoning [28] solve the following objective to construct δ_i that minimize the loss of the surrogate network on clean training data:

$$\min_{\delta \in \Delta} \mathbb{E}_{(x_i, y_i) \sim \mathcal{D}_{\text{train}}} \left[\mathcal{L}(f(x_i + \delta_i), y_i; \theta^*) \right] \tag{2.4}$$

In both error-maximizing and error-minimizing poisoning, the adversary optimizes perturbations δ_i for the training data that respectively maximize or minimize the loss of a surrogate network. Dubbed *unlearnable examples*, images perturbed with error-minimizing noises are a surprisingly good data poisoning attack. A ResNet-18 network trained on a CIFAR-10 [39] sample-wise error-minimizing poison achieves 19.9% final test accuracy, while the class-wise variant achieves 16.4% final test accuracy after 60 epochs of training [28]. The discovery of unlearnable examples also included analysis of an error-maximizing noise optimized using a 20-step PGD attack, but results demonstrated that error-maximizing noise was not as effective in degrading test set performance of a classifier. Fowl et al. [29], however, found that error-maximizing perturbations

can poison a network to achieve 6.25% test accuracy on CIFAR-10 when a 250-step PGD attack is used, suggesting that stronger adversarial attacks could unlock new poisoning capabilities. In Section 3.3, we evaluate whether stronger adversarial attacks are effective at poisoning. Additionally, both error-minimizing and error-maximizing poisons require a surrogate network, from which perturbations are optimized. The optimization can be expensive: crafting the main CIFAR-10 poison from Fowl et al. [29] takes roughly 6 hours on 4 GPUs. In Chapter 4, we develop new poisoning perturbations that do not require access to a surrogate network and can be generated quickly, without the need for backpropagation or a GPU.

There are other notable ways to optimize perturbations for creating unlearnable data. Shen et al. [40] optimize perturbations to explicitly cause gradient vanishing. Feng et al. [41] train an auto-encoder-like network to generate perturbations against a victim classifier. Yuan and Wu [34] use NTKs to understand network predictions and optimize perturbations to produce misclassifications. Shaopeng et al. [42] and Wang et al. [43] propose variants of a tri-level objective and optimize error-minimizing noise for robustness against adversarial training. Peng and Chen [31] propose a class-wise, invertible transformation to control learnability of a dataset for authorized users, while ensuring the data remains unlearnable for other users. The diversity of approaches motivates an exploration of whether learning shortcuts are a commonality that binds them together.

2.2.2 Synthetic Perturbations

Arguably, the simplest way to create unlearnable data is to use class-wise perturbations sampled from a continuous uniform distribution, *i.e.* $\{\delta_i\}_1^K$ where $\delta_i \sim \mathcal{U}[-\epsilon, \epsilon]$. In this case,

the image perturbation δ_i becomes a learning shortcut for class i. When a trained classifier is evaluated on clean data, which does not contain perturbations, it will exhibit random chance test accuracy [28, 33, 44]. On the other hand, if random noise is applied sample-wise, a model trained on this poison can still generalize to the test set [28, 29]. Given their simplicity, random noises for data poisoning have been explored as necessary baselines for unlearnable datasets.

While one can design random perturbations with varying structure, nearly any pattern will work so long as it is applied class-wise. Good examples include the fixed number of uniform patch regions from Yu et al. [33], Regions-16 (See Figure 4.9), or random blurs [36]. Class-wise random perturbations work well in a standard training setup using a RN-18, but their performance varies across architectures and they are rendered ineffective against strong data augmentations like Cutout [45], CutMix [46], and Mixup [47]. In Chapter 4, our autoregressive perturbations degrade test performance more than error-maximizing, error-minimizing, and random poisons on almost every architecture. We also demonstrate that AR perturbations are effective against strong data augmentations and can even mitigate some effects of adversarial training.

2.2.3 Attacks

Unlearnable datasets are not useful if they can be circumvented easily. That is, if the data collector that trains on a poison can still achieve good performance on the clean distribution, then the poison is ineffective. Many attacks have been shown to reduce a poison's effectiveness: adversarial training [28, 35, 43, 48], early-stopping [28, 44, 49], and diluting the poison with clean data [28, 29]. Adversarial training seeks to make models robust to worst-case perturbations by additively applying them during training. Early stopping, on the other hand, seeks to take

advantage of the fact that neural networks learn low-frequency features, which generalize well, early in training. By stopping training early, models can be prevented from overfitting learning shortcuts. Finally, mixing clean data to poisons increases generalization, given that correlations between perturbations and labels are increasingly broken with more clean data.

2.2.4 Analysis

There are a variety of explanations for how unlearnable datasets prevent networks from test set generalization: error-minimizing perturbations cause overfitting [28], error-maximizing noises stimulate learning non-robust features [29], convolutional layers are receptive to autoregressive patterns [32], and more. Different methods arose from different optimization objectives and theory, which lead to slightly different explanations. But Yu et al. [33], find a striking similarity among all the unlearnable datasets they consider: near perfect linear separability of perturbations. They explain that unlearnable datasets cause learning shortcuts due to linear separability of perturbations. In Chapter 4, we find a counterexample, demonstrating that while linear separability of perturbations may certainly be a property that helps unlearnable datasets function, the property is not necessary. Our analysis suggests that even perturbations that are not linearly separable can cause learning shortcuts. In Chapter 3, we will explore general properties that make for effective poisons. We will find that the more quickly perturbations help minimize the training loss (as is typical for learning shortcuts), the more effective the poison is. Whether or not the noise is error-minimizing, training on unlearnable datasets minimizes the loss much faster than clean data [44], suggesting that the resulting dataset is easier. One obvious way to make image classification easy is to include the class label in the image. While unlearnable datasets do not explicitly include

the class label in the image, the perturbations so closely co-occur with the label that it is as if the label were actually part of the image. In general, our work will demonstrate that unlearnable datasets work when simple[1] perturbations are correlated with ground truth labels.

2.3 Learning Shortcuts and Simplicity Bias

Learning shortcuts describe the phenomena that occurs when deep learning models learn simple rules that are not useful for generalization. The term can be considered ill-defined, as deep learning models *optimize* rules that are challenging to comprehend, let alone articulate [50]. For the purpose of this work, and due to our focus on image classification, we will consider poor performance on a particular held-out subset of data as evidence of a learning shortcut. Take the example described in Geirhos et al. [1], where an image classifier cannot correctly assign a class label "cow" to an image of a cow on the beach. Because the classifier cannot generalize to background changes, one may infer that "cow" features (*e.g.* the shape of its head, the positioning of its eyes, the size of its body) are not actually being used for inference. When the held-out test data is wildly different from the training data, this could also be viewed as a distribution shift [51].

Neural networks are known to exhibit a simplicity bias: in the presence of several complex predictive features and a few simple predictive features, neural networks exclusively learn and exploit the simple features at the expense of robustness [52]. Shah et al. [52] note that simplicity bias "is extreme": neural networks choose the simplest feature (like the background) even when all other features have equal predictive power. The authors conclude that this preference results in the susceptibility of neural networks to adversarial perturbations and spurious correlations.

[1] Simple perturbations are defined in Section 4.1

Part I of this dissertation provides further evidence that simplicity bias is the primary mechanism through which imperceptible perturbations control what a neural network learns. Image perturbations, despite being constrained to be small, have an outsized impact on generalization.

2.4 Robustness

A recognition system is robust if it remains accurate despite transformations to the input data. For example, common training time augmentations make classifiers robust to rotations, color jitter, or random crops of an image. Real world scenarios require that models be resilient to inherent noise or variations. Hendrycks and Dietterich [53] argue that recognition systems should be more like humans, robust even to abstract changes in structure and style. They propose a benchmark, *ImageNet-C* which introduces 15 algorithmically generated corruptions, with 5 levels of severity each. After finding that ImageNet models are biased toward texture, Geirhos et al. [54] design Stylized-ImageNet, a stylized version of ImageNet, to learn shape-based representations and improve robustness to *ImageNet-C* distortions. Because image classification datasets typically consist of centered objects in common camera poses, Barbu et al. [55] design the ObjectNet dataset, where objects of interest are positioned and framed in unique ways. Benchmarks are important for improving robustness against diverse forms of distortions and variations in structure and style.

Malicious transformations, like adversarial perturbations, are important for maintaining reliable behavior in critical situations. However, these transformations are less prevalent in the real world due to the high associated cost and the expertise required for their generation [56]. Current methods for maintaining robustness to these worst-case perturbations negatively affect

test accuracy [57, 58].

2.5 Transformers

In recent years, the transformer architecture [59] has redefined what is possible in computer vision [60] and natural language processing [61]. At a high level, transformers are a type of deep neural network consisting of repeated blocks, where each block consists of a self-attention module and a feed-forward module. The self-attention operation can either be bidirectional or causal. Transformers take tokens as input, and tokens are just vectors that store information about the input data.

Bidirectional transformers are also called encoders because a token at position i can attend to every other position j. In contrast, causal transformers, which are typically used in language modeling, allow the token at position i to only attend to tokens at positions $j \leq i$. Causal transformers and are the focus of our study in Chapter 8 and we use the term "transformers" to refer to causal transformers throughout the remainder of this dissertation. Transformers are typically trained using Next Token Prediction (NTP), where given a dataset of sequences, the loss captures how well we predict the next token given previous tokens. Transformers autoregressively predict the next token given only previous tokens.

Crucially, it is not just language that can be modeled as a sequence. Given an image, you can partition pixels into contiguous patches then linearly embed them into tokens [60]. In this way, transformers could be used for image classification and detection [62]. It has turned out that practically every task can be modeled as a sequence: speech recognition [63], text to speech [64], image generation [65], etc. Due to the prevalence of transformers in recent years, part of this

dissertation focuses on understanding what pretrained transformers learn.

2.5.1 Interpretability

Like most neural networks, the algorithms that transformers learn from data are often difficult to understand. Research has focused on the attention sub-module, partly because it is a crucial component of the Transformer and partly because we have anthropomorphized the mechanism. Attention has been described as the network *choosing to look* at different parts of the input data [66]. For example, [66] find attention heads that look at the direct objects of verbs, determiners of nouns, and objects of prepositions with high accuracy. In vision transformers, attention maps have been used for object detection [67] and semantic segmentation [68], despite the network not being explicitly trained for these tasks.

Looking beyond the attention mechanism, the work of Elhage et al. [69] proposes a framework for looking at transformers as compositions of linear operations. The idea is to decompose computation into circuits: groups of attention heads and MLP neurons that implement algorithms. Subsequent work identified a two-head circuit for induction (copying repeated patterns) [70]. In other words, induction heads are attention heads that increase the likelihood of a token "B" given "A B ... A". They learn to repeat arbitrary sequences, which one possible mechanism that contributes to the in-context learning abilities of modern transformer LLMs [8].

2.5.2 Attention Sinks

Despite there being some interpretable attention heads in transformer large language models, it is surprising that a significant number of attention heads attend to semantically irrelevant tokens

[71, 72], function words (*e.g.* , "and" and "of"), or delimiter tokens [73, 74]. The disproportionate attention weight allocated to these irrelevant tokens (typically the first token) has led to them being referred to as attention sinks. However, even in 2019 during the early days of analyzing the BERT model [75], and long before attention sinks became a term, Clark et al. [66] found that "a surprisingly large amount of BERT's attention focuses on the deliminator token [SEP], which we argue is used by the model as a sort of no-op." When one token disproportionately receives a high fraction of attention, it inevitably takes away weight from other token's value states. If the token that received the highest amount of attention also happens to have a near-zero value state, then the attention module will "no-op" by not modifying the hidden states prior to the feedforward module.

Another way to make sense of this behavior is to look at residual networks, of which the transformer is an instance. If it were optimal to skip a block from a ResNet, "it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers" [76]. For the self-attention module to be a no-op, it would need to produce a zero output. One way to do is to attend to a token with a zero value state. Under this view, transformers learn attention sinks from data to effectively skip softmax attention.

Some modern attention variants remove softmax normalization [77, 78]. However, in this dissertation, we use the term "attention" to refer to softmax attention [59]. That is, dot products between queries and keys are scaled with a softmax prior to multiplication by value states.

Part I

Making Unlearnable Datasets using Learning Biases

Chapter 3: Poisons That are Learned Faster are More Effective

Imperceptible poisoning attacks on entire datasets have recently been touted as methods for protecting data privacy. However, among a number of defenses preventing the practical use of these techniques, early-stopping stands out as a simple, yet effective defense. To gauge poisons' vulnerability to early-stopping, we benchmark error-minimizing, error-maximizing, and synthetic poisons in terms of peak test accuracy over 100 epochs and make a number of surprising observations. First, we find that poisons that reach a low training loss faster have lower peak test accuracy. Second, we find that a current state-of-the-art error-maximizing poison is $7\times$ less effective when poison training is stopped at epoch 8. Third, we find that stronger, more transferable adversarial attacks do not make stronger poisons. We advocate for evaluating poisons in terms of peak test accuracy.[1]

Overall, our contributions can be summarized as follows:

- We find that the early epochs of poison training are a good indicator of whether a poison will reach a high test accuracy. There exists a correlation between peak test accuracy and the number of epochs before a threshold loss is reached.

- We demonstrate that final test accuracy does not correlate well with peak test accuracy. We

[1]This chapter follows "Poisons That Are Learned Faster Are More Effective" [44] by Pedro Sandoval-Segura, Vasu Singla, Liam Fowl, Jonas Geiping, Micah Goldblum, David Jacobs, and Tom Goldstein, and was published in the Art of Robustness Workshop at CVPR 2022.

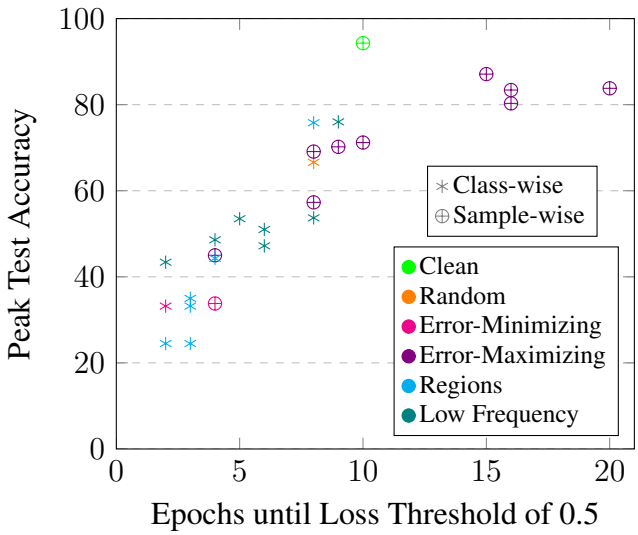


Figure 3.1: Poisons which are learned faster have lower peak test accuracy on CIFAR-10. For every poison, we log both the epoch after which the training loss dips below 0.5 and the peak test accuracy over 100 epochs.

advocate for evaluating poisons based on *peak test accuracy* over final test accuracy.

- We find that adversarial attacks which are stronger or more transferable do not always lead to more effective error-maximizing poisons.

Employing a variety of poisons, our experiments were designed to answer two key questions: Are poisons that are learned faster a better defense against early stopping? and do stronger, more transferable adversarial attacks make more effective poisons?

3.1 Experimental Setup

3.1.1 Optimized Poisons

We craft a number of error-maximizing poisons using the open-source code of Fowl et al. [29]. In particular, we employ targeted 10, 100, and 250-step PGD [22] and MI-FGSM [79] attacks. The poison crafted using a targeted 250-step PGD attack is the main poison published

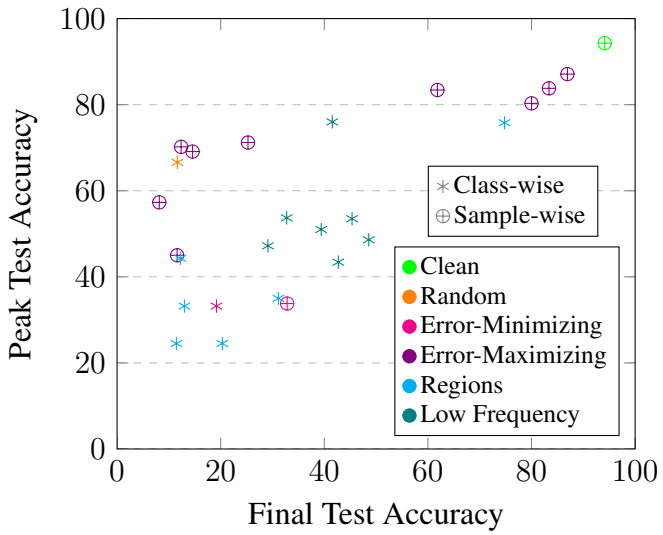


Figure 3.2: There are a number of effective poisons for which final test accuracy doesn't correlate well with peak test accuracy. Effective poisons have both low peak test accuracy and low final test accuracy.

by Fowl et al. [29], where a ResNet-18 trained on the poison achieves a final test set accuracy below 10% on CIFAR-10. Given that Fowl et al. [29] only release the untargeted version of their poison, we crafted the targeted version using their open-source code. Our functional poisons make use of a functional attack, ReColorAdv (C) [80], which can be combined with a spatial attack, StAdv (S) [81] and an additive delta (D) attack like PGD. Following the naming style from Laidlaw and Feizi [80], the **FCSD** poison perturbs images using ReColor, StAdv, and PGD. The functional poisons are the only poisons where perturbations are not constrained to an ℓ_∞ ball around the original image. Functional attacks use different distance measures to quantify perceptual similarity between the clean and the adversarial example.

We also craft two error-minimizing poisons, **SU** and **CU**, using open-source code from Huang et al. [28], one of which is sample-wise and the other class-wise.

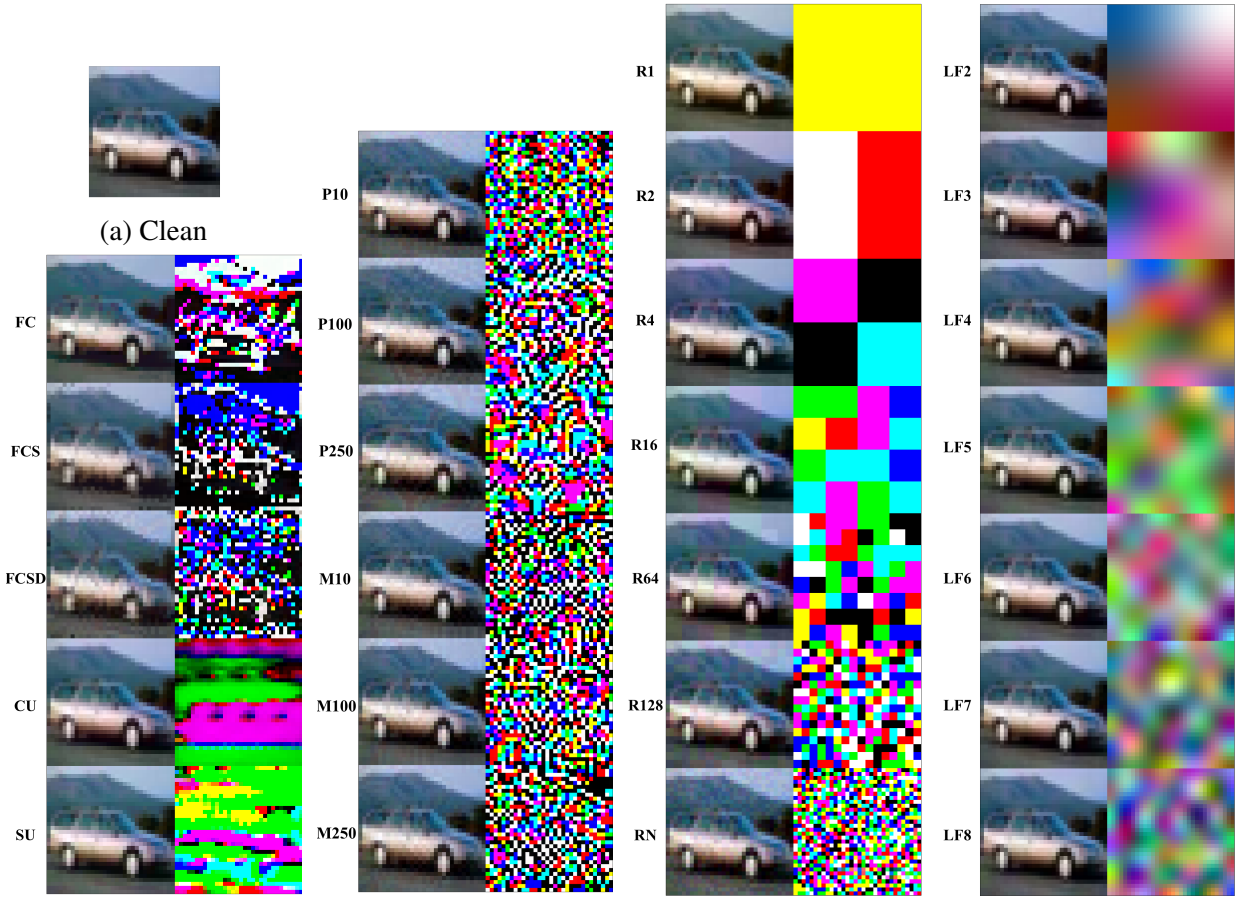


Figure 3.3: A CIFAR-10 image and its corresponding normalized perturbation from different poisons labeled by IDs from Table 3.1. The original clean sample is shown in (a). Error-minimizing and error-maximizing poisons are shown in columns (b) and (c). Synthetic random poisons are shown in columns (d) and (e).

3.1.2 Synthetic Poisons

A quick glance at the normalized error-minimizing and error-maximizing perturbations in Figure 3.3 illustrates the difficulty of understanding why they work. A much simpler noise type to understand is that of synthetic, random perturbations. Our synthetic poisons are randomly generated and are designed to take advantage of two kinds of features that convolutional networks seem to be biased towards: spatially-local [82] and low frequency [83, 84]. Both these synthetic

ID	Detail	Type	Appl.	Peak Acc (↓)	Final Acc (↓)
P10	10-step PGD	Max	S	83.43	61.84
P100	100-step PGD	Max	S	45.02	11.59
P250	250-step PGD [29]	Max	S	57.30	**8.15**
M10	10-step MIFGSM	Max	S	71.18	25.27
M100	100-step MIFGSM	Max	S	69.05	14.58
M250	250-step MIFGSM	Max	S	70.20	12.36
FC	100-step Functional C	Max	S	87.09	86.93
FCS	100-step Functional C+S	Max	S	83.82	83.37
FCSD	100-step Functional C+S+D	Max	S	80.31	80.00
SU	SW-Unlearnable [28]	Min	S	**33.77**	32.83
CU	CW-Unlearnable [28]	Min	C	33.22	19.19
R1	1 region	R	C	75.83	74.79
R2	2 regions	R	C	34.96	31.14
R4	4 regions	R	C	**24.51**	**11.45**
R16	16 regions	R	C	24.54	20.32
R64	64 regions	R	C	33.21	13.00
R128	128 regions	R	C	44.26	12.22
RN	1024 regions	R	C	66.58	11.64
LF2	2x2 DCT	R	C	53.51	45.36
LF3	3x3 DCT	R	C	43.36	42.71
LF4	4x4 DCT	R	C	48.59	48.59
LF5	5x5 DCT	R	C	47.16	29.12
LF6	6x6 DCT	R	C	51.04	39.40
LF7	7x7 DCT	R	C	53.72	32.74
LF8	8x8 DCT	R	C	75.96	41.54

Table 3.1: A summary of all CIFAR-10 poisons considered in this work. Poisons are given an ID for easy reference. Poison perturbations are either Error-Minimizing (Min), Error-Maximizing (Max), or Random (R), and are applied to clean images in a Class-wise (C) or Sample-wise (S) manner. The best class-wise and sample-wise poisons are in bold.

class-wise noises represent a first step in designing poisons which are learned quickly.

Regions noise. To generate a noise with n patch regions, we sample n vectors of size 3 from a Bernoulli distribution. Each vector is then scaled to lie in the range $[-\frac{8}{255}, \frac{8}{255}]$. Finally, the n vectors are repeated along two dimensions to achieve a shape of $32 \times 32 \times 3$. With the exception of the **R2** poison, a Regions noise contains patches of size $\frac{32}{\sqrt{n}} \times \frac{32}{\sqrt{n}}$. Our Regions perturbations are similar to those of Yu et al. [33], but we scale our perturbations to reach the edge of the ϵ-ball.

Low frequency noise. To generate low frequency perturbations, we first sample an $n \times n \times 3$ matrix of Gaussian noise, representing a DCT block. We then append zeros to the matrix along the first two dimensions to achieve a shape of $32 \times 32 \times 3$ and subsequently perform the inverse

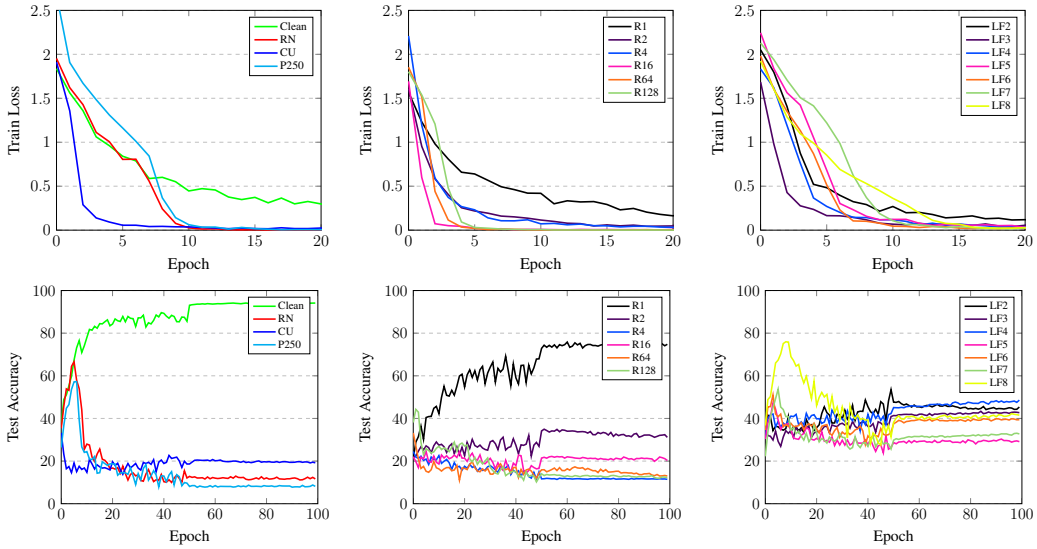


Figure 3.4: Train Loss and Test Accuracy curves when training a ResNet-18 on a variety of poisons. The first column displays curves for a clean training run plus three selected poisons: class-wise random noise (RN), class-wise unlearnable (CU), and 250-step PGD (P250). The second column displays curves for regions poisons. The third column displays curves for low-frequency poisons.

DCT transform. When n is small, the resulting image contains low frequency patterns.

3.1.3 Training Hyperparamters

We train a number of ResNet-18 models on different poisons with cross-entropy loss for 100 epochs using a batch size of 128. For our optimizer, we use SGD with momentum of 0.9 and weight decay of 5×10^{-4}. We use an initial learning rate of 0.1, which decays by a factor of 10 on epoch 50.

3.2 Learning Speed Determines Peak Test Accuracy

What is the best way to imperceptibly modify a training dataset so that a deep neural network trained on the resulting data exhibits the largest generalization gap? This is the question that unlearnable dataset methods seek to answer.

While poison effectiveness is generally measured by the final test set accuracy of a poisoned model, an arbitrarily low final test set accuracy is not effective if the victim could simply stop training early and obtain better test set performance. Thus, we define peak test accuracy as the highest test set performance over 100 epochs, and evaluate our poisons from Table 3.1 on this metric. As we show in Figure 3.2, there are a number of poisons which reach low final test accuracy, but high peak test accuracy.

To define the speed at which a poison is learned, we arbitrarily choose a loss threshold of 0.5 and take note of the first epoch when the training loss is less than 0.5. This is a reasonable loss threshold given that nearly 85% of poisons we tested reach a cross-entropy loss of 0.5 by epoch 10. By taking note of the epoch after which a model reaches this threshold, we obtain a proxy for the ease with which a poison is learned.

For every poison, we hence obtain both an epoch until the loss threshold is reached and the overall peak test accuracy during training. We plot our results in Figure 3.1 and observe that poisons which are learned faster reach lower peak test accuracy, and are thus better defended against early-stopping. We provide training loss and validation accuracy curves in Figure 3.4. Our version of the sample-wise adversarial poison from Fowl et al. [29], which we label **P250**, reaches a final test set accuracy of 8.15%, but reaches a peak test accuracy of 57.3% as shown in Figure 3.4 — the poison is $7\times$ less effective when we stop training at epoch 8! This serves as an example that poisons should be evaluated by their worst-case performance. The correlation between how quickly a poison is learned and its peak test accuracy suggests we should strive to minimize the loss early in training.

Our synthetic regions poisons and low-frequency poisons are a first step at designing perturbations which are learned quickly by convolutional DNNs like ResNet-18. Attempting

Poison	Victim Networks					Poison Detail	
	ResNet-18	VGG-19	MobileNet	GoogLeNet	Avg (↑)	Peak Acc (↓)	Final Acc (↓)
RN	8.35	6.28	23.50	10.43	12.14	66.58	11.64
P10	14.24	7.55	34.55	13.00	17.34	83.43	61.84
P100	83.90	65.55	90.03	81.57	80.26	**45.02**	11.59
P250	95.55	84.58	95.73	90.93	**91.70**	57.30	**8.15**
M10	38.40	20.21	46.56	29.35	33.63	71.18	25.27
M100	53.18	22.79	46.75	43.12	41.46	69.05	14.58
M250	48.70	20.65	46.71	36.42	38.12	70.20	12.36
FC	94.95	38.88	53.88	50.85	59.64	87.09	86.93
FCS	99.97	73.18	76.23	88.69	84.52	83.82	83.37
FCSD	100.00	83.53	86.24	94.74	**91.13**	80.31	80.00

Table 3.2: Attack transferability of different error-maximizing CIFAR-10 poisons against victim models of varying architecture. Peak and Final accuracy (%) are for a ResNet-18 trained on the poison. Values within 1% of the best are in bold.

to take advantage of a CNN's bias for spatially-local image features, our regions poisons have patches of the same color. Taking advantage of a DNN's bias for low-frequency patterns, our low frequency poisons span a spectrum of complexity. Training curves from Figure 3.4 demonstrate that as the number of regions increases, poisons are generally learned more quickly, but there is a sweet spot for the number of regions (**R4** and **R16** have particularly low peak accuracy and final test accuracy). Compared to regions poisons, low frequency poisons perform more poorly overall. Interestingly, as the number of frequencies increases from **LF2** to **LF8**, the poisons appear to be learned more slowly, and the corresponding test accuracy curves show higher peak accuracies. For example, **LF7** reaches a peak accuracy of 53.72% while **LF8** reaches 75.96%. Further work is needed to develop a poison which can induce lower than **R4**'s 24.51% peak test accuracy on CIFAR-10.

3.3 Stronger Adversarial Attacks Do Not Make Stronger Poisons

Both error-maximizing noise and error-minimizing noises were evaluated in Huang et al. [28], but their proposed error-minimizing sample-wise noise produced lower clean test set accuracy across the entire training process compared to other all other poisons. To optimize their error-maximizing noise, they used a 20-step PGD attack.

Despite the apparent success of error-minimizing noise over error-maximizing noise, Fowl et al. [29] found a way to bring error-maximizing noise back into the spotlight: using a stronger 250-step PGD attack [29]. Why did a 250-step attack behave so differently as a poison, when compared to the 20-step PGD attack? The findings of Fowl et al. [29] could be used as an indication that stronger adversarial attacks are the key to improving error-maximizing poisoning performance. Rather than continue to increase the number of attack steps, we opt to investigate whether adversarial attack transferability is correlated with poisoning performance.

Adversarial examples are crafted by optimizing an objective which seeks to maximize the network's prediction error. While many adversarial attacks operate under an additive threat model, where the adversary is allowed to add an imperceptible perturbation vector to a clean image, spatial [81] and functional [80] threat models have also been explored. Functional adversarial attacks introduced a novel class of threat models which, at the time, produced the highest attack success rates even after adversarial training. The ReColorAdv [80] attack uses a parameterized function to map each color in an image to a new color in the adversarial example. Laidlaw and Feizi [80] also explore combining this functional threat model with spatial and additive threat models, resulting in stronger, more transferable attacks. To evaluate whether stronger, more transferable attacks are effective poisons, we conduct an analysis of adversarial attack

transferability across a range of error-maximizing attacks in Table 3.1.

We consider four DNN architectures as victim networks: a ResNet-18 [76], VGG-19 [85], MobileNet [86], and GoogLeNet [87], which we train on CIFAR-10 for 200 epochs. Recall that sample-wise error-maximizing poisons are constructed by performing an adversarial attack on each clean image from the training set. Thus, attack transferability is measured by computing the fraction of examples in the poison which are misclassified by the victim network. We present attack transferability results alongside the corresponding poison's peak and final test accuracy in Table 3.2.

An adversarial attack can be considered stronger than another if it uses more attack steps or if it is more transferable to other DNNs of differing architectures. In terms of peak test accuracy, increasing the number of attack steps for a PGD or MI-FGSM attack does not improve peak test accuracy. For example, **P100** achieves a peak test accuracy of 45.02%, while the more transferable **P250** poison achieves 57.30%. Stronger, more transferable adversarial attacks do not necessarily degrade test accuracy either. The case of **FCSD** is a striking example: the adversarial attack is one of the most transferable across the four victim networks, yet performs very poorly as a poison.

3.4 Conclusion

In this chapter, we made a number of observations which should be taken into account when designing poisons for the purpose of data privacy protection. Motivated by defending against the mitigating effect of early-stopping, we found that poison training exhibits a correlation between how quickly a model reaches a low loss threshold and the peak test accuracy overall. This leads

us to propose that poisons should be learned quickly to ensure peak test accuracy remains low throughout training and user privacy is meaningfully protected.

To further understand this phenomenon, we craft two kinds of synthetic poisons aimed at taking advantage of CNN biases. But while random, spatially-local regions poisons are learned quickly, they suffer from being slightly perceptible. Low frequency perturbations, on the other hand, do not achieve low test accuracy.

Finally, our experiments suggest that adversarial attacks which are more transferable or stronger in terms of steps do not lead to effective error-maximizing poisons. Due to high peak test accuracy, we caution against error-maximizing noises for data poisoning.

Chapter 4: Autoregressive Perturbations for Data Poisoning

This chapter builds on Chapter 3, where we found that effective perturbations are those that a network fits quickly. Here, we show how to certify that property before training. Our method also provides new insight into why data poisoning works.

We introduce autoregressive (AR) data poisoning for degrading overall performance of neural networks on clean data. The perturbations that we additively apply to clean data are generated by AR processes that are data and architecture-independent. An AR(p) process is a Markov chain, where each new element is a linear combination of p previous ones, plus noise. This means AR perturbations are cheap to generate, not requiring any optimization or backpropagation through network parameters. AR perturbations are generic; the same set of AR processes can be re-used to generate diverse perturbations for different image sizes and new datasets, unlike other poisoning methods which need to train a surrogate network on the target dataset before crafting perturbations.[1]

Our method also provides new insight into why data poisoning works. We work on top of the result that effective poisons are typically easy to learn [44] and construct AR perturbations which are separable by a manually-specified CNN. Working under the intuition that highly separable perturbations should be easily learned, we use the manual specification of parameters as

[1]This chapter follows "Autoregressive Perturbations for Data Poisoning" [32] by Pedro Sandoval-Segura, Vasu Singla, Jonas Geiping, Micah Goldblum, Tom Goldstein, and David Jacobs, and was published at NeurIPS 2022.

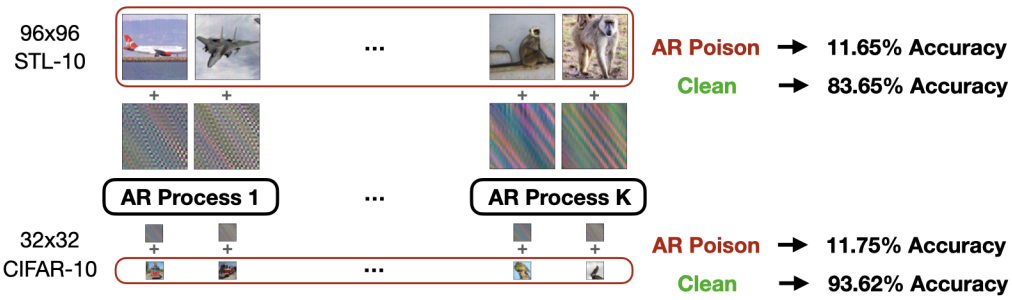


Figure 4.1: The same set of AR processes can generate perturbations for different kinds of data without access to a surrogate network or other images in a dataset. A ResNet-18 trained on these AR poisons is unable to generalize to clean test data, even with the help of strong data augmentations.

a way of demonstrating that our AR perturbations are easily separable. Our manually-specified CNN makes use of what we call AR filters, which are attuned to detect noise from a specific AR process. AR poisoning's effectiveness is competitive or better than error-maximizing, error-minimizing, and random noise poisoning across a range of architectures, datasets, and common defenses. AR poisoning represents a paradigm shift for what a successful indiscriminate poisoning attack looks like, and raises the question of whether strong indiscriminate poisons need to be generated by surrogate networks for a given dataset.

4.1 Why Autoregressive Perturbations?

Perturbations that are easy to learn have been shown to be more effective at data poisoning [44]. Intuitively, a signal that is easily interpolated by a network will be quickly identified and used as a "shortcut," whereas complex and unpredictable patterns may not be learned until after a network has already extracted useful content-based features [52]. Thus, we seek imperceptible perturbations that are easy to learn. We propose a simple hypothesis: if there exists a simple CNN that can classify autoregressive signals perfectly, then these signals will be easy to learn.

The signals can then be applied to clean images and serve as a shortcut for learning by commonly-used CNNs.

Autoregressive perturbations, despite looking visually complex, are actually very simple. To demonstrate their separability, we manually specify the parameters of a simple CNN that classifies AR perturbations perfectly by using AR filters. In the following, we prove AR filters satisfy an important property.

Lemma 4.1.1 *Given an AR perturbation δ, generated from an AR(p) with coefficients $\beta_1, ..., \beta_p$, there exists a linear, shift invariant filter where the cross-correlation operator produces a zero response.*

Proof Consider any size $p+1$ window of the AR perturbation δ where, by definition, the last entry follows Eq. (4.5) directly: $[\mathbf{x}_{t-p}, \mathbf{x}_{t-(p-1)}, ..., \mathbf{x}_{t-1}, \mathbf{x}_t]$. We construct a size $p+1$ filter where the elements are $\beta_p, ..., \beta_1$ and the last entry of the filter is -1. We call this filter an AR filter. Notice that the dot product of a window of δ with this filter is:

$$[\mathbf{x}_{t-p}, \mathbf{x}_{t-(p-1)}, ..., \mathbf{x}_{t-1}, \mathbf{x}_t] \cdot [\beta_p, \beta_{p-1}, ..., \beta_1, -1] \tag{4.1}$$

$$= \beta_p \mathbf{x}_{t-p} + \beta_{p-1} \mathbf{x}_{t-(p-1)} + ... + \beta_1 \mathbf{x}_{t-1} - \mathbf{x}_t \tag{4.2}$$

$$= \beta_p \mathbf{x}_{t-p} + \beta_{p-1} \mathbf{x}_{t-(p-1)} + ... + \beta_1 \mathbf{x}_{t-1} - (\beta_1 \mathbf{x}_{t-1} + ... + \beta_{p-1} \mathbf{x}_{t-(p-1)} + \beta_p \mathbf{x}_{t-p}) \tag{4.3}$$

$$= 0$$

which implies:

$$\delta \star [\beta_1, ..., \beta_p, -1] = \vec{0} \tag{4.4}$$

where $\star$ denotes the cross-correlation operator.

The construction of an AR filter that produces a zero response for any noise generated from the corresponding AR process is useful because we can construct a CNN which makes use of solely these AR filters to classify signals. That is, given any AR perturbation, the AR filter with the zero response correctly designates the AR process from which the perturbation was generated. We verify this claim in Section 4.3 by specifying the 2-layer CNN that can perfectly classify AR perturbations.

Crucially, we are not interested in learning classes of AR signals. Rather, we are interested in how quickly a model can learn classes of clean data *perturbed* by AR signals. Nevertheless, the characterization of our AR perturbations as easy to learn, demonstrated by the manual specification of a 2-layer CNN, is certainly an indication that, when applied to clean data, AR perturbations can serve as bait for CNNs. Our experiments will seek to answer the following question: If we perturb each sample in the training dataset with an imperceptible, yet easily learned AR perturbation, can we induce a learning "shortcut" that minimizes the training loss but prevents generalization?

4.2 Generating Autoregressive Perturbations

Autoregressive (AR) perturbations have a particularly useful structure where local regions throughout the perturbation are Markovian, exposing a linear dependence on neighboring pixels [88]. This property is critical as it allows for a particular filter to perfectly detect noise from a specific AR process, indicating the noise is simple and potentially easily learned.

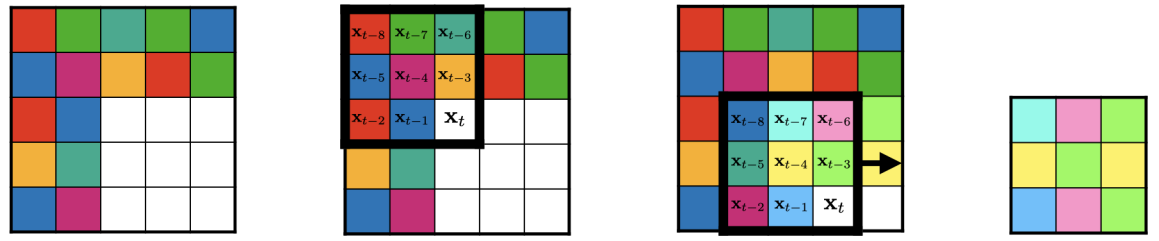


Figure 4.2: To generate 2D noise for a single channel using a 3×3 sliding window, we first sample Gaussian noise for the first 2 columns and rows. Then, we use an AR(8) process within a sliding window to compute the next value. Finally, we crop and scale the perturbation to be within our ℓ_2 constraint.

We develop a sample-wise poison where clean images are perturbed using additive noise. For each x_i in the clean training dataset, our algorithm crafts a δ_i, where $\|\delta_i\|_2 \leq \epsilon$, so that the resulting poison image is $x_i' = x_i + \delta_i$. The novelty of our method is in how we find and use autoregressive (AR) processes to generate δ_i. In the following, let $\mathbf{x}_t$ refer to the t^{th} entry within a sliding window of δ_i.

An autoregressive (AR) process models the conditional mean of $\mathbf{x}_t$, as a function of past observations $\mathbf{x}_{t-1}, \mathbf{x}_{t-2}, ..., \mathbf{x}_{t-p}$ in the following way:

$$\mathbf{x}_t = \beta_1 \mathbf{x}_{t-1} + \beta_2 \mathbf{x}_{t-2} + ... + \beta_p \mathbf{x}_{t-p} + \epsilon_t \tag{4.5}$$

where ϵ_t is an uncorrelated process with mean zero and β_i are the AR process coefficients. For simplicity, we set $\epsilon_t = 0$ in this chapter. An AR process that depends on p past observations is called an AR model of degree p, denoted AR(p). For any AR(p) process, we can construct a size $p + 1$ filter where the elements are $\beta_p, ..., \beta_1$ and the last entry of the filter is -1. This filter produces a zero response for any signal generated by the AR process with coefficients $\beta_p, ..., \beta_1$. We refer to this filter as an AR filter, the utility of which is explained in Section 4.1.

Suppose we have a K class classification problem of $H \times W \times C$ dimensional images. For each class label y_i, we construct a set A_{y_i} of AR processes, one for each of the C channels. For each of the C channels, we will be applying an AR process from A_{y_i} inside a $V \times V$ sliding window. Naturally, using an AR process requires initial observations, so we populate the perturbation vector δ_i with Gaussian noise for the first $V-1$ columns and rows. The $V \times V$ sliding window starts at the top left corner of δ_i. Within this sliding window, we apply the AR($V^2 - 1$) process: the first $V^2 - 1$ entries in the sliding window are considered previously generates (or randomly initialized) entries in the 2D array δ_i, and the V^{th} entry is computed by Eq. 4.5. The window is slid left to right, top to bottom until the first channel of δ_i is filled. We then proceed to use the next AR($V^2 - 1$) process in A_{y_i} for the remaining $C - 1$ channels. Finally, we discard the random Gaussian rows and columns used for initialization, and scale δ_i to be of size ϵ in the ℓ_2-norm. Note that this sliding window procedure resembles that of a convolution. That is by design, and explained in Section 4.1. A high-level overview of this algorithm is illustrated in Figure 4.2. While we describe our use of AR processes on C-channel images, our method could, in principle, be applied to data other than images. Note that these AR perturbations are fast to generate, do not require a pre-trained surrogate model, and can be generated *independently from the data.*

Additional Details In addition to normalizing and scaling AR perturbation after generation, we also find that cropping out the first few columns and rows, where the Gaussian start signal was generated (see Figure 4.2), leads to more effective AR perturbations. We tried cropping only the Gaussian start signal, and cropping the start signal plus additional columns and rows. We found that cropping the start signal plus two extra columns and rows worked best. The AR perturbations

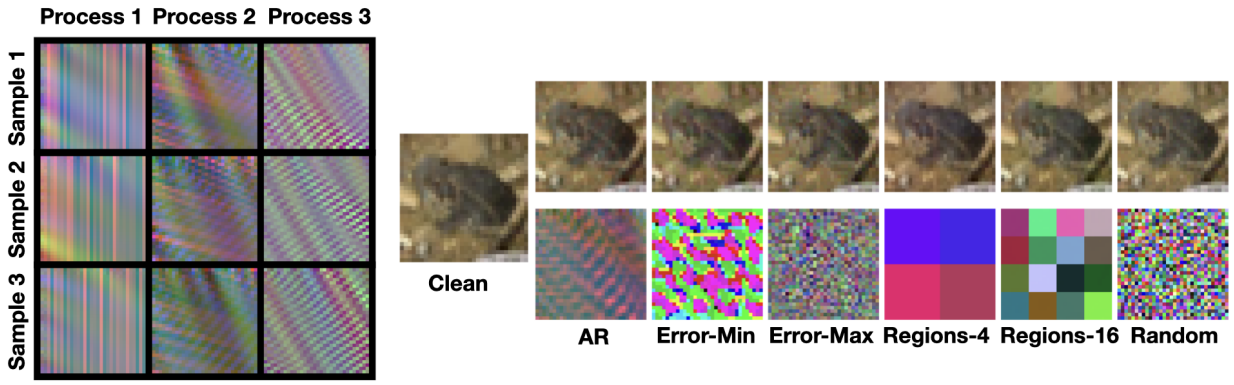


Figure 4.3: **Left**: Normalized samples of perturbations generated by 3 AR processes. **Right**: Poisoned images and the corresponding perturbation for a randomly selected CIFAR-10 image.

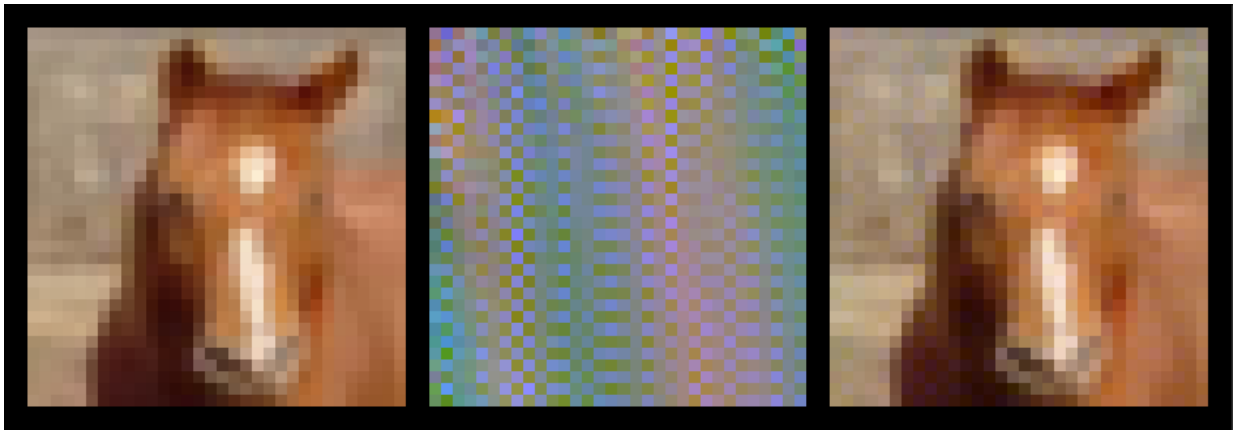


Figure 4.4: A closeup of a CIFAR-10 image from our AR poison. From left to right, we display the clean image, the normalized perturbation, and the perturbed image. Looking closely at the background of the horse, one can make out the checkered blue pattern of this AR perturbation.

are also less perceptible this way, given that the Gaussian noise starting signal has been removed. For 32×32 dimensional images, we generate AR perturbations of size 36×36 and crop the first 4 rows and columns. For 96×96 dimensional images, we generate AR perturbations of size 100×100 and crop the first 4 rows and columns.

4.3 A Two-layer, Manually-specified CNN Can Classify AR Perturbations

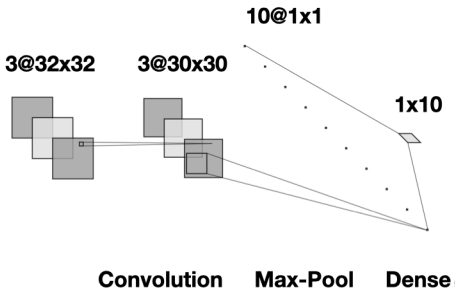


Figure 4.5: A two-layer CNN can perfectly classify AR perturbations without any training. We specify each parameter manually.

Assume AR perturbations are expected to be of size 32×32, as in CIFAR-10. To illustrate the ease with which a set of AR perturbations can be classified, we construct a set of 10 AR(8) processes $A = \vec{\beta_1}, ..., \vec{\beta_{10}}$, where $\vec{\beta_i} = [\beta_{i,1}, ..., \beta_{i,8}]$ is a vector of AR(8) coefficients. Our network consists of 2 learnable layers: a convolutional layer and a fully connected layer. We specify the parameters of each layer in the forward pass order below:

1. Convolutional Layer: 10 AR filters of size 3×3. No bias vector. ReLU activation on output. The i^{th} AR filter is meant to produce a zero response (See Lemma 4.1.1) on noise

generated from AR process with coefficients $\vec{\beta}_i$:

$$f_i = \begin{bmatrix} \beta_{i,8} & \beta_{i,7} & \beta_{i,6} \\ \beta_{i,5} & \beta_{i,4} & \beta_{i,3} \\ \beta_{i,2} & \beta_{i,1} & -1 \end{bmatrix}$$

2. Max-Pool layer: Kernel size of 30×30, the same dimensions as the output of the convolutional layer.

3. Linear layer: We set $W = -I$ and $b = \vec{1}$, where $\vec{1}$ is a vector of all ones. Softmax activation on output.

To classify AR perturbation from 10 classes, this network outputs a probability distribution over 10 classes. This simple CNN works by assigning a high value to the i^{th} logit when the i^{th} AR filter produces a zero response. For example, when an AR filter outputs a zero response, the activation after ReLU and Max-Pool is still 0. The linear layer will then add a bias of 1, so the resulting i^{th} logit will be 1. Other logits correspond to AR filters which do not match the AR process, and output some nonnegative value, after ReLU and Max-Pool. The subsequent linear layer subtracts the nonnegative activation from 1, resulting in a logit value that is smaller than 1. In this way, the softmax activation assigns a high probability to the i^{th} class.

To confirm our network design, we generate $5,000$ AR perturbations per AR process, and use the network defined above to classify the noises. The manually-specified network achieves *perfect accuracy without any training*. We use this result as a demonstration that AR perturbations are separable. Note that for other simple, linearly separable image datasets it is non-trivial to manually specify a CNN for perfect accuracy. Again, our motivation in exploring easily classified

noises is that there is substantial work suggesting dataset separability is key to crafting good poisons.

4.4 Finding AR Process Coefficients

We generate AR processes using a random search that promotes diversity. We generate processes one-at-a-time by starting with a random Gaussian vector of coefficients. We then scale the coefficients so that they sum to one. We then append a -1 to the end of the coefficients to produce the associated AR filter, and convolve this filter with previously generated perturbations. We use the norms of the resulting convolution outputs as a measure of similarity between processes. If the minimum of these norms is below a cutoff T, then we deem the AR process too coherent with previously generated perturbations – the coefficients are discarded and we try again with a different random vector.

Once the AR process coefficients are identified for a class, we use them to produce a perturbation δ_i for each image in the class. This perturbation is scaled to be exactly of size ϵ in the ℓ_2-norm. To level the playing field among all poisoning methods, we measure all perturbations using an ℓ_2 norm in this chapter.

Algorithm 1 Random Search for AR Coefficients

Input: Number of classes, K
Input: Minimum response threshold, T
Output: Set of AR process coefficients, A

1: $A \leftarrow \{\}$
2: **while** $|A| \neq K$ **do**
3: $\quad \vec{\beta} \leftarrow \mathcal{N}(0, 1)$
4: $\quad \vec{\beta} \leftarrow \frac{\vec{\beta}}{\sum_j \beta_j}$
5: $\quad \delta \leftarrow \text{ARGenerate}(\vec{\beta})$
6: $\quad$ **if** δ is stable **then**
7: $\quad\quad$ Let m store the min response so far.
8: $\quad\quad$ **for** i in 1,...,$|A|$ **do**
9: $\quad\quad\quad f \leftarrow \text{ARFilter}(A_i)$
10: $\quad\quad\quad r \leftarrow \text{ConvResponse}(\delta, f)$
11: $\quad\quad\quad m \leftarrow \min(m, r)$
12: $\quad\quad$ **end for**
13: $\quad\quad$ **if** $m \geq T$ **then**
14: $\quad\quad\quad A \leftarrow A \cup \{\beta_i\}$
15: $\quad\quad$ **end if**
16: $\quad$ **end if**
17: **end while**
18: **return** A

Algorithm 2 Generate C-channel AR Perturbation

Input: Image and label, (x_i, y_i), where $x_i \in \mathbb{R}^{H \times W \times C}$ and $y_i \in \{1, ..., K\}$
Input: Size of sliding window, V
Input: Set of AR($V^2 - 1$) processes, A
Input: Size of perturturbation ϵ in ℓ_2
Output: Poisoned image, x_i'

1: Sample $\mathcal{N}(0, 1)$ for first $V - 1$ rows and columns of δ_i
2: **for** k in $1, ..., C$ **do**
3: $\quad \beta_1, ..., \beta_{V^2-1} \leftarrow A^k_{y_i}$
4: $\quad \delta^k_i \leftarrow \text{ARGenerate}(\beta_1, ..., \beta_{V^2-1})$
5: **end for**
6: $\delta_i \leftarrow \epsilon \frac{\delta_i}{\|\delta_i\|_2}$
7: $x_i' \leftarrow x_i + \delta_i$
8: **return** x_i'

We define important functions and variables of Algorithm 1 and Algorithm 2:

- $\vec{\beta}$ is a vector where the entries are the AR process coefficients $\beta_1, ..., \beta_p$.
- To check whether δ is stable, as in Algorithm 1, Line 6, we check whether the AR process diverges under a few starting signals. To do this, we check that the ℓ_2-norm of δ is not large, infinity, or NaN, under 3 different Gaussian starting signals. If the AR process diverges on any starting signal, then we say δ is not stable, and the Algorithm 1 continues by sampling a new set of AR process coefficients.
- $\text{ARGenerate}(\vec{\beta})$ takes a set of AR process coefficients and uses the AR process within a sliding window which goes left to right, top to bottom. The first $V - 1$ rows and columns

of δ_i are sampled from a Gaussian. The output of $\mathrm{ARGenerate}(\vec{\beta})$ is a two-dimensional array. The procedure is described in Section 4.2 and illustrated in Figure 4.2. Algorithm 2 produces the entire C-channel AR perturbation.

- $\mathrm{ARFilter}(A_i)$ takes a set of AR process coefficients and returns the corresponding AR filter, where the entries are the AR coefficients followed by a -1. An AR filter is constructed in Section 4.2 and used in the construction of our simple AR noise classifier in Section 4.3.

- $\mathrm{ConvResponse}(\delta, f)$ takes a signal and filter as input. It measures a filter f's response on δ by performing 2D convolution, ReLU, and sum.

Additional Details While there exist conditions for AR coefficients for which the AR process converges [88, 89], we found that randomly searching for coefficients was the simplest way to collect a large number of suitable AR processes. Additionally, using convergence conditions leads to AR processes which converge too quickly. For our purposes, converging too quickly is not desirable because most of the values in our AR perturbation would be zero.

Instead, we conduct a random search for K sets of converging AR coefficients by randomly sampling values from a Gaussian distribution and ensuring the resulting process is stable; *i.e.* does not diverge. We find that the likelihood of a convergent AR process is much higher when the coefficients are normalized to sum to 1. Normalizing the coefficients (Algorithm 1, Line 4) is also important because we want AR filters to produce a zero response to uniform regions of an image. If AR coefficients sum to 1 and the final entry of the AR filter is -1, then the 2D cross-correlation operator produces a zero response if pixels in a local window are approximately equal.

We optimize so that every AR process is sufficiently different from other processes already

included in our growing set. We do this by measuring the response of AR filters from AR processes already in our set to noise generated by the currently sampled AR process. It may be possible to optimize a different criteria, but in our implementation, $\mathrm{ConvResponse}(\delta, f)$ measures a filter's response by performing 2D convolution, ReLU, and sum. We fully outline this random search for K sets of AR process coefficients in Algorithm 1.

We conduct a search for $K = 10$ sets of AR coefficients using Algorithm 1 with a threshold $T = 10$. This search took roughly 11 hours running on a single CPU. For CIFAR-100, we conduct a search for $K = 100$ sets of AR coefficients with $T = 3$ due to the larger number of classes. This search took roughly 4 hours running on a single GPU. Because Algorithm 1 samples coefficients independently on every iteration, significant speedups can likely be achieved using parallelized code.

4.5 Experimental Setup

We demonstrate the generality of AR poisoning by creating poisons across four datasets, including different image sizes and number of classes. Notably, we use the *same* set of AR processes to poison SVHN [90], STL-10 [91], and CIFAR-10 [39] since all of these datasets are 10 class classification problems. We demonstrate that despite the victim's choice of network architecture, AR poisons can degrade a network's accuracy on clean test data. We show that while strong data augmentations are an effective defense against all poisons we consider, AR poisoning is largely resistant. Adversarial training and diluting the poison with clean data remain strong defenses, but our AR poisoning method is competitive with other poisons we consider. All experiments follow the same general pattern: we train a network on a poisoned dataset and

then evaluate the trained network's performance on clean test data. A poison is effective if it can cause the trained network to have poor test accuracy on clean data, so lower numbers are better throughout our results.

4.5.1 Datasets and Models

SVHN [90], CIFAR-10, and CIFAR-100 [39] poisons considered in this chapter contain perturbations of size $\epsilon = 1$ in ℓ_2, unless stated otherwise. For STL-10 [91], all poisons use perturbations of size $\epsilon = 3$ in ℓ_2 due to the larger size of STL-10 images. In all cases, perturbations are normalized and scaled to be of size ϵ in ℓ_2, are additively applied to clean data, and are subsequently clamped to be in image space. Dataset details can be found in Section 4.11.2. A sampling of poison images and their corresponding normalized perturbation can be found in Section 4.11.1. In our results, class-wise poisons are marked with $\circ$ and sample-wise poisons are marked with $\bullet$.

Error-Max and Error-Min Perturbations To generate error-maximizing poisons, we use the open-source implementation of Fowl et al. [29]. In particular, we use a 250-step ℓ_2 PGD attack to optimize Eq. (2.3). To generate error-minimizing poisons, we use the open-source implementation of [28], where a 20-step ℓ_2 PGD attack is used to optimize Eq. (2.4). For error-minimizing poisoning, we find that moving in ℓ_2 normalized gradient directions is ineffective at reaching the required universal stop error [28], so we move in signed gradient directions instead (as is done for ℓ_∞ PGD attacks).

Regions-4 and Regions-16 Perturbations Synthetic, random noises are also dataset and network independent. Thus, to demonstrate the strength of our method, we include three class-wise

random noises in our experiments. To generate what we a call a Regions-p noise, we follow [33, 44]: we sample p RGB vectors of size 3 from a Gaussian distribution and repeat each vector along height and width dimensions, resulting in a grid-like pattern of p uniform cells or regions. Assuming a square image of side length L, a Regions-p noise contains patches of size $\frac{L}{\sqrt{p}} \times \frac{L}{\sqrt{p}}$.

Random Noise Perturbations We also consider a class-wise random noise poison, where perturbations for each class are sampled from a Gaussian distribution.

4.5.2 Training Hyperparameters

We train a number of ResNet-18 (RN-18) [76] models on different poisons with cross-entropy loss for 100 epochs using a batch size of 128. For our optimizer, we use SGD with momentum of 0.9 and weight decay of 5×10^{-4}. We use an initial learning rate of 0.1, which decays by a factor of 10 on epoch 50. In Table 4.2, we use use the same settings with different network architectures.

4.6 AR Perturbations are Dataset and Architecture Independent

Unlike error-maximizing and error-minimizing poisons, AR poisons are not dataset-specific. One cannot simply take the perturbations from an error-maximizing or error-minimizing poison and apply the same perturbations to images of another dataset. Perturbations optimized using PGD are known to be relevant features, necessary for classification [29, 92]. Additionally, for both these methods, a crafting network trained on clean data is needed to produce reasonable gradient information. In contrast, AR perturbations are generated from dataset-independent AR processes. The same set of AR processes can be used to generate the same kinds of noise for

Table 4.1: **Dataset Independence.** AR noises are effective across a variety of datasets. For SVHN, STL-10, and CIFAR-10, we use the same 10 AR processes to generate sample-wise noises. We display clean test accuracy of RN-18 when trained using standard augmentations on different poisons. Class-wise poisons are marked with ○ and sample-wise poisons are marked with ●.

	SVHN	STL-10	CIFAR-10	CIFAR-100
Clean	96.40	83.65	93.62	75.30
● Error-Max [29]	80.80	17.26	26.94	4.87
● Error-Min [28]	96.82	80.71	16.84	74.58
○ Regions-4	9.80	41.36	20.75	9.14
○ Regions-16	**6.39**	31.21	15.75	**2.99**
○ Random Noise	9.68	72.07	18.45	73.90
● Autoregressive (Ours)	6.77	**11.65**	**11.75**	4.24

images of new datasets. Building from this insight, one could potentially collect a large set of K AR processes to perturb any dataset of K or fewer classes, further showing the generality of our method.

In Table 4.1, we use the same 10 AR processes to generate noise for images of SVHN, STL-10, and CIFAR-10. AR poisons are, in all cases, either competitive or the most effective poison – a poison-trained RN-18 reaches nearly chance accuracy on STL-10 and CIFAR-10, and being the second-best on SVHN and CIFAR-100. The generality of AR perturbations to different kinds of datasets suggests that AR poisoning induces the most easily learned correlation between samples and their corresponding label.

We also evaluate the effectiveness of our AR poisons when different architectures are used for training. Recall that error-maximizing and error-minimizing poisoning use a crafting network to optimize the input perturbations. Because it may be possible that these noises are specific to the network architecture, we perform an evaluation of test set accuracy on CIFAR-10 after poison training VGG-19 [85], GoogLeNet [87], MobileNet [86], EfficientNet [93], DenseNet [94], and

Table 4.2: **Architecture Independence.** CIFAR-10 test accuracy for a variety of model architectures trained on different poisons. Error-Max and Error-Min poisons are crafted using a RN-18.

	RN-18	VGG-19	GoogLeNet	MobileNet	EfficientNet	DenseNet	ViT
• Error-Max [29]	16.84	20.34	15.31	15.38	16.21	18.02	41.70
• Error-Min [28]	26.94	22.39	32.18	21.36	23.86	28.21	40.95
○ Regions-4	20.75	25.39	25.25	22.23	18.64	24.21	35.18
○ Regions-16	15.75	19.86	12.97	19.67	16.94	24.16	25.49
○ Random Noise	18.45	81.98	12.53	46.61	86.89	**9.96**	74.47
• AR (Ours)	**11.75**	**12.35**	**9.24**	**15.17**	**13.47**	14.90	**19.66**

ViT [60]. Our ViT uses a patch size of 4. In Table 4.2, we show that Error-Max and Error-Min poisons generalize relatively well across a range of related CNNs, but struggle with ViT, which is a transformer architecture. In contrast, our AR poison is effective across all CNN architectures and is the most effective poison against ViT. Our AR poison is much more effective over other poisons in almost all cases, achieving improvements over the next best poison of 4% on RN-18, 5.8% on ViT, and 7.5% on GoogLeNet. The design of AR perturbations is meant to target the convolution operation, so it is surprising to see a transformer network be adversely affected. We believe our AR poison is particularly effective on GoogLeNet due to the presence of Inception modules that incorporate convolutions using various filter sizes. While our AR perturbations are generated using a 3×3 window, the use of various filter sizes may exaggerate their separability.

4.7 AR Perturbations Work Against Common Defenses

4.7.1 Against Data Augmentations and Smaller Perturbations

Our poisoning method relies on imperceptible AR perturbations, so it is conceivable that one could modify the data to prevent the learning of these perturbations. One way of modifying data is by using data augmentation strategies during training. In addition to standard augmentations

Table 4.3: **Strong Data Augmentations.** CIFAR-10 test accuracy of RN-18 when training using standard augmentations plus Cutout, CutMix, or Mixup on different poisons, where clean images are perturbed within an ℓ_2 ball of size ϵ.

		Standard Aug	+Cutout	+CutMix	+Mixup
• Error-Max [29]	$\epsilon = 0.5$	25.04	25.38	29.72	38.07
• Error-Min [28]		48.07	44.97	53.77	53.81
○ Regions-4		57.48	67.47	67.72	66.80
○ Regions-16		47.35	39.64	32.67	49.02
○ Random Noise		93.76	93.66	91.80	94.16
• Autoregressive (Ours)		**14.28**	**12.36**	**18.02**	**14.59**
• Error-Max [29]	$\epsilon = 1$	16.84	18.86	21.45	30.52
• Error-Min [28]		26.94	29.38	25.04	43.36
○ Regions-4		20.75	21.89	28.61	40.60
○ Regions-16		15.75	19.61	16.67	20.81
○ Random Noise		18.45	12.61	12.63	23.64
• Autoregressive (Ours)		**11.75**	**11.90**	**11.23**	**11.40**

like random crops and horizontal flips, we benchmark our AR poison against stronger augmentations like Cutout [45], CutMix [46], and Mixup [47] in Table 4.3. Generally, Mixup seems to be the most effective at disabling poisons. A RN-18 trained on a poison using standard augmentations plus Mixup can achieve a boosts in test set performance of 13.68% on Error-Max, 16.42% on Error-Min, 19.85% on Regions-4, 5.05% on Regions-16, and 5.19% on Random Noise. However, a RN-18 poison-trained on our AR poison ($\epsilon = 1$) using standard augmentations plus Cutout, CutMix, or Mixup cannot achieve *any* boost in test set performance.

We also present results for poisons using perturbations of size $\epsilon = 0.5$ to explore just how small perturbations can be made while still maintaining poisoning performance. Under standard augmentations, going from larger to smaller perturbations ($\epsilon = 1$ to $\epsilon = 0.5$), poison effectiveness drops by 8.2% for Error-Max, 21.13% for Error-Min, 36.73% for Regions-4, and 31.6% for Regions-16. Our AR poison achieves the smallest drop in effectiveness: only 2.53%. Random noise can no longer be considered a poison at $\epsilon = 0.5$ – it completely breaks for small

perturbations. Under *all* strong data augmentation strategies at $\epsilon = 0.5$, AR poisoning dominates. For example, under Mixup, the best runner-up poison is Error-Max with an effectiveness that is more than 23% lower than AR. Unlike all other poisons, AR poisoning is exceptionally effective for small perturbations.

Note that in all three augmentation strategies pixels are either dropped or scaled. Our method is unaffected by these augmentation strategies, unlike error-maximizing, error-minimizing, and other randomly noise poisons. Scaling an AR perturbation does not affect how the corresponding matching AR filter will respond,[2] and thus, the patterns remain highly separable regardless of perturbation size. Additionally, AR filters contain values which sum to 0, so uniform regions of an image also produce a zero response.

4.7.2 Against Adversarial Training

Adversarial training has also been shown to be an effective counter strategy against ℓ_p-norm constrained data poisons [28, 29, 30, 35]. Using adversarial training, a model trained on the poisoned data can achieve nearly the same performance as training on clean data [42]. However, adversarial training is computationally more expensive than standard training and leads to a decrease in test accuracy [22, 57] when the perturbation radius, ρ_a, of the adversary is large. In Table 4.4, we include adversarial training results on clean data to outline this trade-off where training at large ρ_a comes at the cost of test accuracy. A recent line of work has therefore focused on developing better data poisoning methods that are robust against adversarial training [42, 43] at larger adversarial training radius ρ_a.

In Table 4.4, we compare performance of different poisons against adversarial training. We

[2] See condition outlined in Lemma 4.1.1.

Table 4.4: **Adversarial Training.** CIFAR-10 test accuracy after adversarially training with different radii ρ_a. Top row shows performance of adversarial training on clean data. AR poisons remain effective for small ρ_a.

	ρ_a			
	0.125	0.25	0.50	0.75
Clean Data	87.07	84.75	81.19	77.01
• Error-Max [29]	$33.30_{\pm 0.14}$	$72.27_{\pm 2.18}$	$81.15_{\pm 3.58}$	$78.73_{\pm 4.20}$
• Error-Min [28]	$70.66_{\pm 0.41}$	$84.80_{\pm 2.38}$	$83.04_{\pm 3.24}$	$79.11_{\pm 3.46}$
○ Regions-4	$75.05_{\pm 0.35}$	$81.23_{\pm 0.11}$	$79.71_{\pm 0.05}$	$76.47_{\pm 0.34}$
○ Regions-16	$47.99_{\pm 0.25}$	$71.43_{\pm 0.17}$	$80.47_{\pm 0.10}$	$76.65_{\pm 0.07}$
○ Random Noise	$86.31_{\pm 0.42}$	$84.17_{\pm 0.20}$	$\mathbf{80.11_{\pm 0.06}}$	$\mathbf{76.26_{\pm 0.07}}$
• Autoregressive (Ours)	$\mathbf{33.22_{\pm 0.77}}$	$\mathbf{57.08_{\pm 0.75}}$	$81.27_{\pm 2.61}$	$79.07_{\pm 3.47}$

perform ℓ_2 adversarial training with different perturbation radii, ρ_a, using a 7-step PGD attack with a step-size of $\frac{\rho_a}{4}$. We report error-bars by training three independent models for each run. We also show the performance of adversarial training on clean data. Data poisoning methods are fragile to adversarial training even when the training radius ρ_a is smaller than poisoning radius ϵ [42, 43]. It is desirable for poisons to remain effective for larger ρ_a, because the trade-off between standard test accuracy and robust test accuracy would be exaggerated further. As shown in the Table 4.4, when the adversarial training radius ρ_a increases, the poisons are gradually rendered ineffective. All poisons are nearly ineffective at $\rho_a = 0.5$. Our proposed AR perturbations remain more effective at smaller radius, *i.e.* $\rho_a = 0.125$ and $\rho_a = 0.25$ compared to all other poisons.

4.7.3 Against Mixing with Clean Data

Consider the scenario when not all the data can be poisoned. This setup is practical because, to a practitioner coming into control of poisoned data, additional clean data may be available through other sources. Therefore, it is common to evaluate poisoning performance using smaller proportions of randomly selected poison training samples [28, 29, 42]. A poison

Table 4.5: **Mixing Poisons with Clean Data.** CIFAR-10 test accuracy when a proportion of clean data is used in addition to a poison. Top row shows test accuracy when training on only the clean proportion of the data; *i.e.* no poisoned data is used.

	Clean Proportion				
	40%	30%	20%	10%	5%
Clean Only	90.84	89.92	87.90	81.01	74.97
• Error-Max [28]	$87.83_{\pm 0.74}$	$86.83_{\pm 0.48}$	$84.70_{\pm 0.61}$	$81.63_{\pm 0.63}$	$76.48_{\pm 1.72}$
• Error-Min [29]	$88.32_{\pm 1.57}$	$87.23_{\pm 0.84}$	$84.56_{\pm 0.88}$	$78.76_{\pm 1.83}$	$67.82_{\pm 1.92}$
○ Regions-4	$88.94_{\pm 0.85}$	$86.75_{\pm 0.86}$	$83.52_{\pm 0.20}$	$78.23_{\pm 0.97}$	$70.19_{\pm 3.16}$
○ Regions-16	$88.03_{\pm 0.57}$	$86.23_{\pm 0.68}$	$\mathbf{83.01_{\pm 0.48}}$	$76.52_{\pm 0.91}$	$67.24_{\pm 1.72}$
○ Random Noise	$\mathbf{86.40_{\pm 1.24}}$	$86.99_{\pm 0.19}$	$84.98_{\pm 1.85}$	$78.08_{\pm 0.94}$	$70.69_{\pm 0.87}$
• AR (Ours)	$87.63_{\pm 0.68}$	$\mathbf{85.62_{\pm 0.62}}$	$83.28_{\pm 0.90}$	$\mathbf{76.13_{\pm 2.34}}$	$\mathbf{62.69_{\pm 5.58}}$

can be considered effective if the addition of poisoned data hurts test accuracy compared to training on only the clean data. In Table 4.5, we evaluate the effectiveness of poisons using different proportions of clean and poisoned data. The top row of Table 4.5 shows test accuracy after training on only the subset of clean data, with no poisoned data present. We report error-bars by training four independent models for each run. Our AR poisons remain effective compared to other poisons even when clean data is mixed in. AR poisons are much more effective when a small portion of the data is clean. For example, when 5% of data is clean, a model achieves $\sim 75\%$ accuracy when training on only the clean proportion, but using an additional 95% of AR data leads to a $\sim 9\%$ decrease in test set generalization. Our results on clean data demonstrate that AR poisoned data is worse than useless for training a network, and a practitioner with access to the data would be better off not using it.

4.8 AR Poisoning in ℓ_∞

Autoregressive perturbations can be projected onto any ℓ_p-norm ball, including ℓ_∞. We initially measured the ℓ_2-norm because AR perturbations may have single entries which are high

and violate a strict ℓ_∞ constraint. To demonstrate that AR poisoning *can* work in the ℓ_∞-norm constrained setting, we train a RN-18 on an AR poison with perturbations satisfying $\|\delta\|_\infty = \frac{8}{255}$, and report CIFAR-10 test accuracy in Table 4.6.

Table 4.6: **Poisoining in ℓ_∞.** CIFAR-10 test accuracy of an RN-18 trained on an AR poison using standard augmentations plus Cutout, CutMix, or Mixup.

	Standard Aug	+Cutout	+CutMix	+Mixup
Autoregressive (Ours)	20.49	26.93	17.08	15.22

Importantly, how one fits an AR perturbation δ within the constraint that $\|\delta\|_\infty \leq \epsilon = \frac{8}{255}$ affects performance. In this experiment, we simply scale δ by $\frac{\epsilon}{\|\delta\|_\infty}$, which may be suboptimal. Clipping values or taking the scaled sign of δ would make the perturbation no longer autoregressive. AR perturbations could likely be optimized to perform better in ℓ_∞ by making modifications to how AR coefficients are compared in Algorithm 1.

4.9 Effect of Optimizer Hyperparameters

In Table 4.7, we report mean CIFAR-10 test accuracy over the 9 combinations of optimizers and learning rates. Our Autoregressive poison remains nearly unaffected by the choice of hyperparameters, and performs better than all other poisons.

Table 4.7: **Effect of Optimizer and LR.** Mean CIFAR-10 test accuracy of an RN-18 trained on our AR poison, over 9 optimizer and LR combinations.

Autoregressive (Ours)	Random Noise	Regions-4	Regions-16	Error-Max.	Error-Min
$12.23_{\pm 1.22}$	$33.39_{\pm 31.24}$	$22.89_{\pm 7.6}$	$18.73_{\pm 5.52}$	$16.6_{\pm 2.8}$	$22.96_{\pm 7.16}$

To evaluate the effect of optimizer and learning rate on our poisoning results, we train an RN-18 on poisons using different optimizers. We consider 3 optimizers: SGD, SGD+Momentum

($\beta = 0.9$) and Adam ($\beta_1 = 0.9$, $\beta_2 = 0.999$). We also consider 3 different learning-rate schedules: cosine learning-rate, single-step decay (at epoch 50) and multi-step decay (at epochs 50 and 75) with decay factor of 0.1. There are a total of 9 optimizer and learning rate combinations.

4.10 Conclusion

Using the intuition that simple noises are easily learned, we proposed the design of AR perturbations, noises that are so simple they can be perfectly classified by a 2-layer CNN where all parameters are manually-specified. We demonstrate that these AR perturbations are immediately useful and make effective poisons for the purpose of preventing a network from generalizing to the clean test distribution. Unlike other effective poisoning techniques that optimize error-maximizing or error-minimizing noises, AR poisoning does not need access to a broader dataset or surrogate network parameters. We are able to use the same set of 10 AR processes to generate imperceptible noises able to degrade the test performance of networks trained on three different 10 class datasets. Unlike randomly generated poisons, AR poisons are more potent when training using a new network architecture or strong data augmentations like Cutout, CutMix, and Mixup. Against defenses like adversarial training, AR poisoning is competitive or among the best for a range of attack radii. Finally, we demonstrated that AR poisoned data is worse than useless when it is mixed with clean data, reducing likelihood that a practitioner would want to include AR poisoned data in their training dataset.

There are a number of modifications which could potentially make for more effective AR perturbations, the exploration of which we leave to future work. For example, the size of the sliding window or the order of the AR process could potentially lead to perturbation values

which converge more slowly. A higher order AR process could also be applied along the channel dimension.

Additionally, poisoning a K-class classification dataset requires K AR processes. The random search for AR coefficients is certainly an area of improvement; there may exist more efficient ways of finding effective AR coefficients. Nevertheless, having found a set of K AR processes, Table 4.1 is evidence that they can be reused for any other classification dataset with K or fewer classes.

4.11 Appendix

4.11.1 Samples of CIFAR-10 Poisons

We plot a random sample of 30 poison images from each CIFAR-10 poison used in this work. In each plot, we also illustrate the corresponding normalized perturbation for each image. The Error-Max poison is shown in Figure 4.6, Error-Min poison in Figure 4.7, Regions-4 poison in Figure 4.8, Regions-16 poison in Figure 4.9, and our AR poison is shown in Figure 4.10.

4.11.2 Dataset Details

SVHN [90] contains 10 classes, where 73257 images are for training and 26032 images are for testing. STL-10 [91] conatins 10 classes, where 5000 are for training and 8000 are for testing. Both CIFAR-10 and CIFAR-100 [39] contain 50000 images for training and 10000 images for testing. Images of CIFAR-10 belong to one of 10, whereas images from CIFAR-100 belong to one of 100 classes. SVHN, CIFAR-10, and CIFAR-100 contain images of size $32 \times 32 \times 3$; STL-10 contains images of size $96 \times 96 \times 3$.

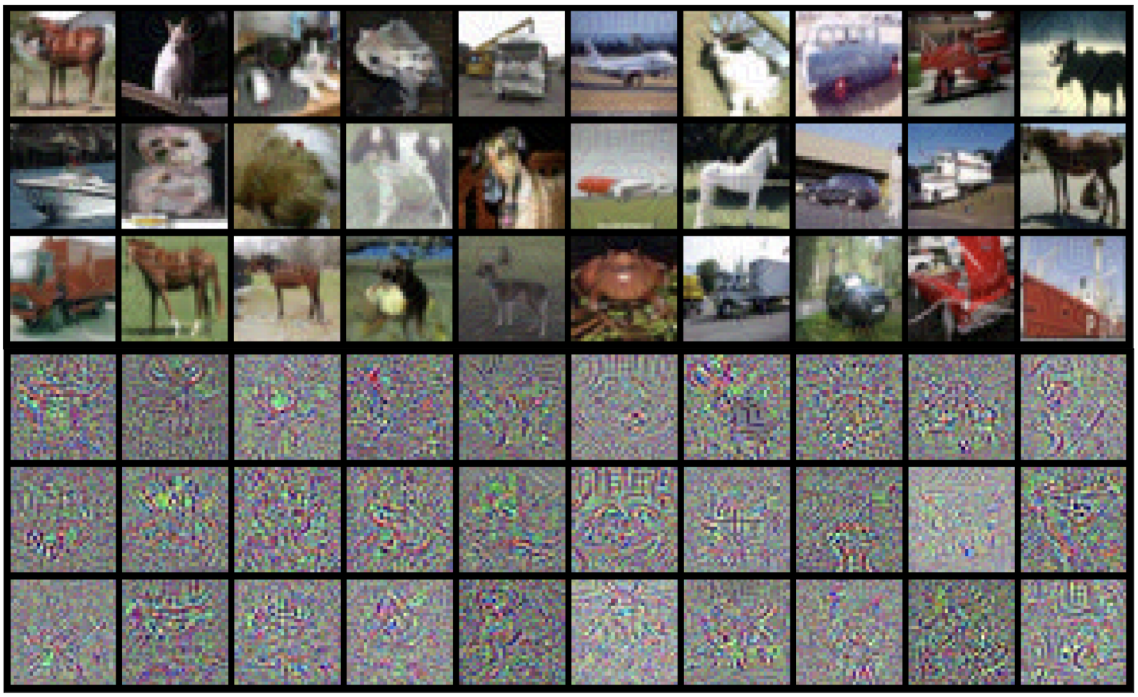


Figure 4.6: **Error-Max Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

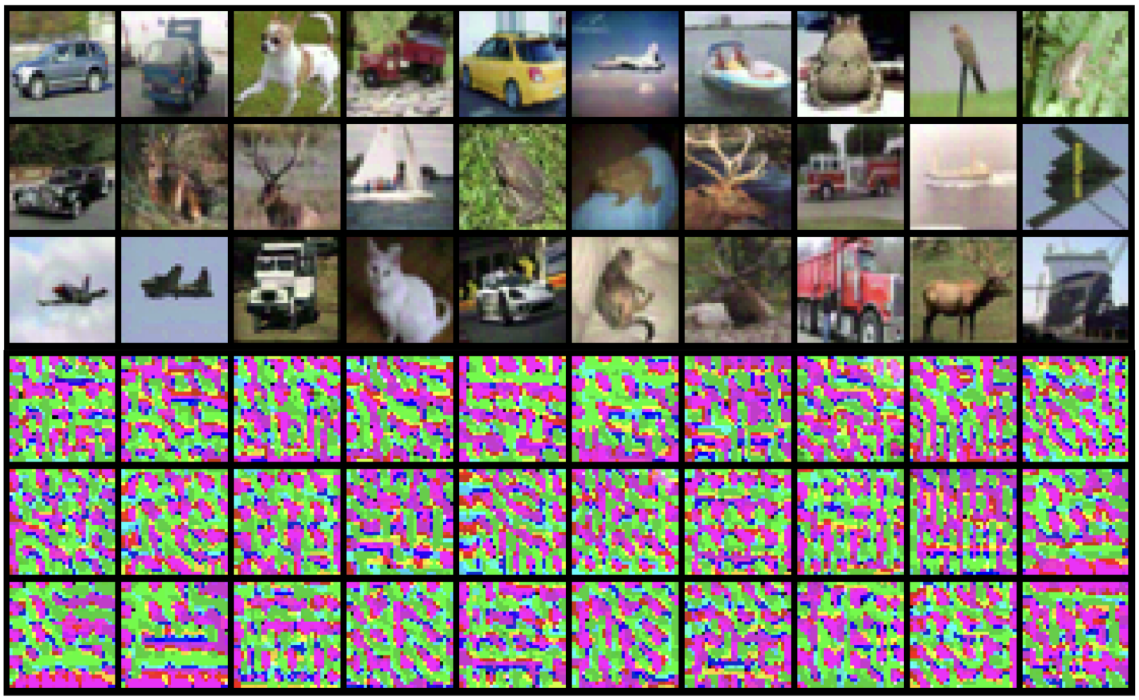


Figure 4.7: **Error-Min Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

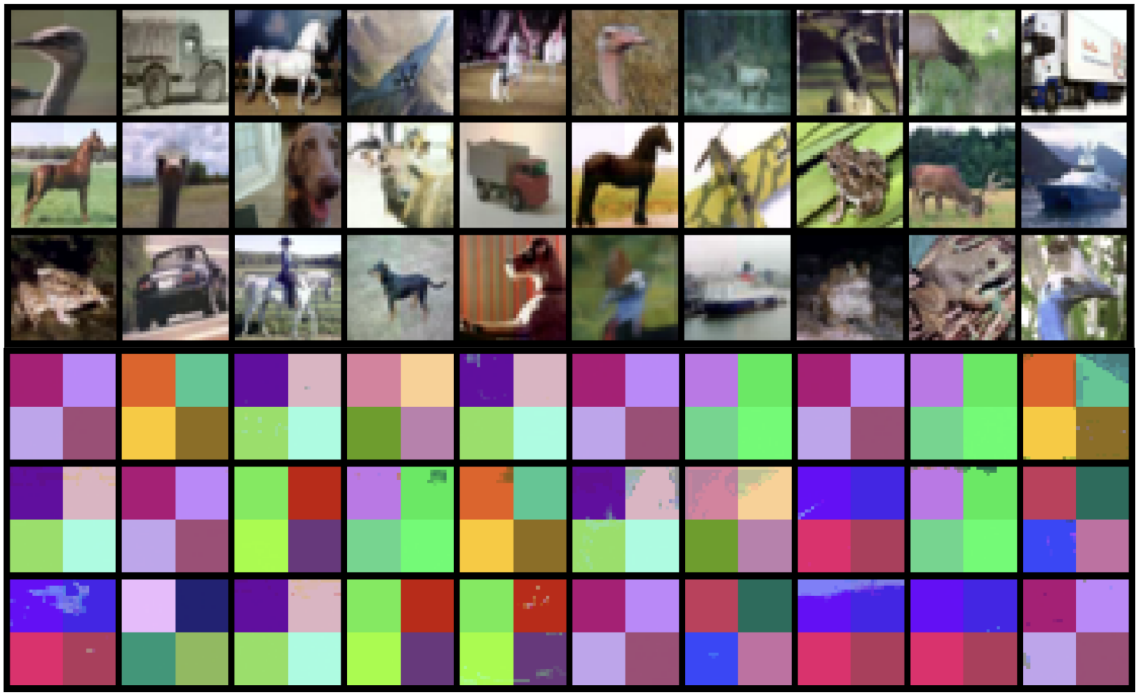


Figure 4.8: **Regions-4 Poison.**. A random sample of 30 poison images and their corresponding normalized perturbation.



Figure 4.9: **Regions-16 Poison.** A random sample of 30 poison images and their corresponding normalized perturbation.

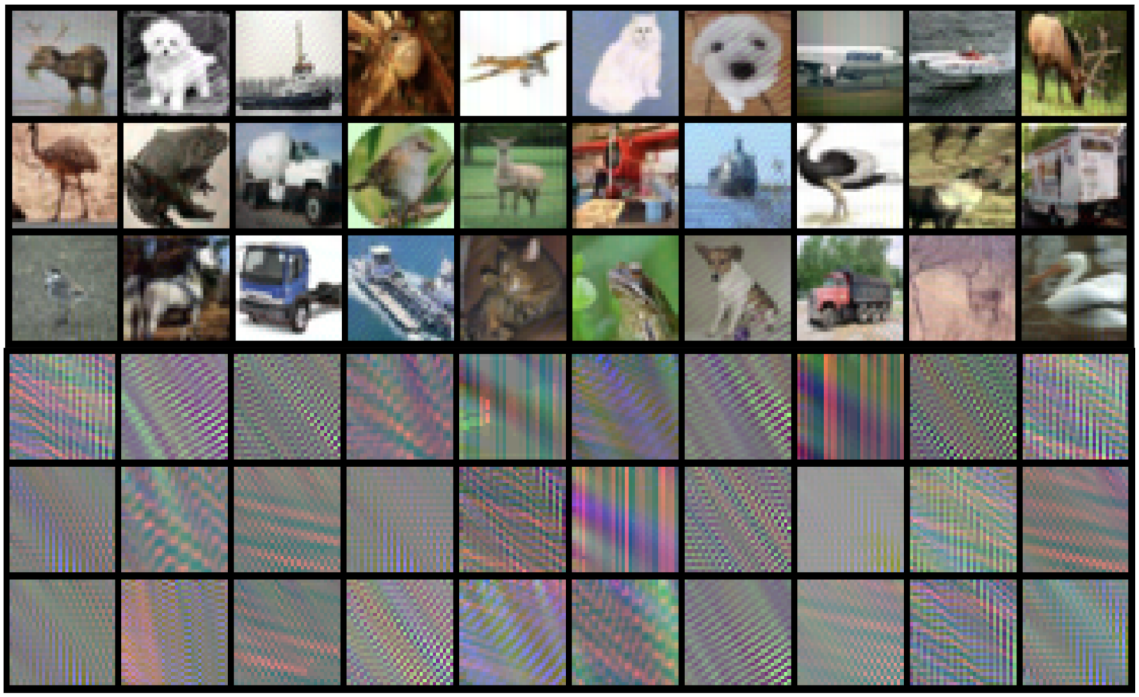


Figure 4.10: **AR Poison (Ours).** A random sample of 30 poison images and their corresponding normalized perturbation.

Part II

The Limits of Perturbation-based Protection

Chapter 5: What Can We Learn From Unlearnable Datasets?

In this chapter, we analyze properties of unlearnable dataset methods in order to assess their future viability and security promises. Currently, unlearnable dataset methods are unlikely to be used for safeguarding public data because they require perturbing a large portion (*e.g.* more than 50%) of the training set [28, 32] and adversarial training is a principled attack [35]. Additionally, published data is immutable and must withstand current and future attacks [95]. While there are a number of reasons why these unlearnable dataset methods are unlikely to be employed, our results shed light on privacy implications and challenge common hypotheses about how they work. We make several findings by analyzing a number of unlearnable datasets developed from diverse objectives and theory.[1] In particular,

- We demonstrate that, in many cases, neural networks can learn generalizable features from unlearnable datasets. Our results imply that while resulting test accuracy may be low, DNNs may still learn useful features from protected data.

- We challenge the common belief that unlearnable datasets work due to linearly separable perturbations. We construct a counterexample, suggesting that there is room for new unlearnable dataset methods that evade common attacks.

[1] This chapter follows "What Can We Learn from Unlearnable Datasets?" [37] by Pedro Sandoval-Segura, Vasu Singla, Jonas Geiping, Micah Goldblum, and Tom Goldstein, and was published at NeurIPS 2023.

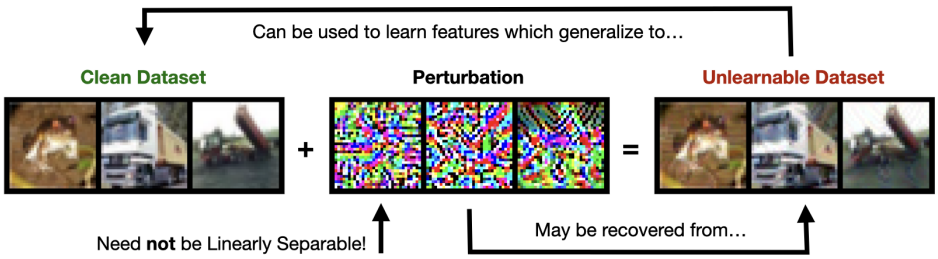


Figure 5.1: We study unlearnable datasets constructed from additive perturbations. Although unlearnable datasets are created to prevent the exploitation of the clean dataset, we show that unlearnable datasets can be used to learn features which generalize to clean test data. We demonstrate that unlearnable datasets can be created without linearly separable perturbations. And we develop a novel method for recovering class-wise, linearly separable perturbations from unlearnable datasets.

- Inspired by our results on linear separability of perturbations, we present a new Orthogonal Projection attack which allows learning from unlearnable datasets perturbed by class-wise, linearly separable perturbations. Our attack demonstrates that class-wise applied perturbations do not protect data from being learned. Our attack is often more effective than adversarial training, at a fraction of the computational cost.

5.1 Experimental Setup

5.1.1 Datasets and Models

Datasets For all of our experiments in this chapter, we make use of open-source unlearnable datasets: From *Unlearnable Examples* [28], we use their sample-wise and class-wise poisons. From *Adversarial Poisoning* [29], we use their targeted PGD attack poison. From *Neural Tangent Generalization Attacks (NTGA)* [34], *Robust Unlearnable Examples* [42], and *Linearly Separable Perturbations (LSP)* [33], we use their main poison. From *Autoregressive Poisoning (AR)* [32] introduced in the previous chapter, we use the dataset with perturbations of size 1 in ℓ_2. For *One-*

pixel Shortcut [38], we use their OPS and CIFAR-10-S poisons, but we refer to their CIFAR-10-S poison as *OPS+EM* for ease of attribution. We also include two class-wise *Regions-4* and *Random Noise* poisons which contain randomly sampled perturbations, following [44]. To poison a K-class dataset in a class-wise manner, we need K perturbation vectors. For *Regions-4*, each perturbation is independently created by sampling 4 vectors of size 3 from a Bernoulli distribution. Each vector is then scaled to lie in the range $[-\frac{8}{255}, \frac{8}{255}]$. Finally, each of the 4 vectors are repeated along height and width dimensions to create patches of size 16×16, which are then arranged side by side to achieve a shape of 32×32. For *Random Noise* perturbations, we sample an i.i.d. vector, one entry per pixel, from a Bernoulli and scale perturbations to fit the imperceptibility constraint. See Section 5.4.4 for image samples for all unlearnable datasets we consider.

Results in this section are for CIFAR-10 [39], SVHN [90], CIFAR-100 [39], and ImageNet [96] Subset. Unlearnable datasets constructed with class-wise perturbations are prefixed with $\circ$, otherwise the perturbations were added sample-wise. For the imperceptibility constraint, the *AR* (ℓ_2) dataset contains perturbations δ of size $\|\delta\|_2 \leq 1$. *LSP* has perturbations of size $\|\delta\|_2 \leq \epsilon'\sqrt{d}$, where d is the image dimension and $\epsilon' = \frac{6}{255}$. *OPS* and *OPS+EM* contain unbounded perturbations. All other datasets contain perturbations of size $\|\delta\|_\infty \leq \frac{8}{255}$.

Models We primarily use the ResNet-18 (RN-18) [76] architecture for training on the above datasets, but we also include results for additional network architectures including VGG-16 [85], GoogleNet [87], and ViT [60].

5.1.2 Training Hyperparameters

Section 5.2 We train a number of RN-18 [76] models on different unlearnable datasets with cross-entropy loss for 60 epochs using a batch size of 128. We save checkpoints at every epoch of training. For our optimizer, we use SGD with momentum of 0.9 and weight decay of 5×10^{-4}. We use an initial learning rate of 0.1 which decays using a cosine annealing schedule.

For training a new classification layer on feature extractor checkpoints, we use $5,000$ random clean images from the original training set data. Note that $5,000$ images is 10% of CIFAR-10 and CIFAR-100, but 3.9% of Imagenet subset, etc. Following Kirichenko et al. [97], using the feature extractor, we extract embeddings from this clean subset of data and preprocess the embeddings to have mean zero and unit standard deviation. To retrain the last layer, we use the logistic regression implementation from `scikit-learn`.

Section 5.3 For every unlearnable dataset, we first gather the set of perturbations by subtracting the clean image from the perturbed image. We zero-one normalize the perturbations before training a linear layer using L-BFGS [98] for 500 steps using a learning rate of 0.5.

Section 5.4 For CIFAR-10, we train the logistic regression model for 20 epochs using SGD with an initial learning rate of 0.1, which decays by a factor of 10 on epochs 5 and 8 (at epochs which are 0.5 and 0.75 through training). For CIFAR-100, we train the logistic regression model for 60 epochs due to the higher number of classes. After we orthogonally project the unlearnable data using the optimized weights, we train networks using the hyperparameters from checkpoint-training of Section 5.2.

Table 5.1: **Generalizable features can be learned from unlearnable datasets.** While published unlearnable datasets cause DNNs to train to low test accuracy, the features learned by DNNs can be reweighted using DFR to high test accuracy in many cases. For each unlearnable dataset, we report test accuracy and test loss for the best performing checkpoint after DFR. In gray, we indicate test accuracy improvement/deterioration over DFR on a randomly initialized RN-18. DFR uses $5,000$ clean samples for finetuning. For the majority of unlearnable datasets, peak performance seems to occur early in training. High DFR Test Accuracy, and Low DFR Test Loss, indicates that useful features are learned during training.

Training Data	Max DFR Test Accuracy	Min DFR Test Loss
None	35.97	2.379
Unlearnable Examples [28]	39.56 (+3.59)	1.798
Adversarial Poisoning [29]	69.99 (+34.02)	1.036
AR (ℓ_2) [32]	58.73 (+22.76)	1.531
NTGA [34]	57.12 (+21.15)	1.391
Robust Unlearnable [42]	41.02 (+5.05)	1.790
LSP [33]	43.31 (+7.34)	1.675
OPS+EM [38]	38.98 (+3.01)	1.869
∘ OPS [38]	47.70 (+11.73)	1.697
∘ Unlearnable Examples [28]	34.09 (-1.88)	2.037
∘ Regions-4 [44]	48.09 (+12.12)	1.706
∘ Random Noise	66.79 (+30.82)	1.352

5.2 Learning Useful Features from Unlearnable Datasets

Unlearnable datasets are meant to be unlearnable; models trained on unlearnable datasets should not be capable of generalizing to clean test data. Presumably, the modified, unlearnable dataset protects the clean version of the dataset from unauthorized use in this fashion. But by reweighting deep features, we find that DNNs are still able to recover some generalizable features from unlearnable datasets.

DFR Method The technique of training a new linear classification layer on top of a fixed feature extractor is known as Deep Feature Reweighting (DFR) [97] and it is a method for learning from data with spurious correlations, where predictive features in the train distribution are not present

in the test distribution. We borrow the DFR method to better understand the utility of features learned during poison training. The higher the test accuracy after DFR, the more likely it is that the model has learned features present in the original clean data. If DFR works well on poison-trained networks, this would suggest that even poisoned weights contain a semblance of relevant information for generalization.

To evaluate the extent that generalizable features are learned, we start by saving network weights at every epoch of poison training. In the context of image classification, network weights consist of a feature extractor followed by a fully-connected classification layer mapping feature vectors to class logits. Next, for each checkpoint, we utilize a random subset of $5,000$ clean CIFAR-10 training samples (10% of the original training set) to train a new classification head (and keep the feature extractor weights fixed). Finally, we plot test accuracy in Figure 5.2 and evaluate the maximum CIFAR-10 test accuracy achieved through DFR for each dataset and report results in Table 5.1. As a baseline, we train a classification head on a randomly initialized RN-18 feature extractor and find that these random features can be reweighted to achieve 35.97% test accuracy.

DFR Results We find that, to different extents, *DNNs do learn generalizable features* from unlearnable datasets. Surprisingly, on *Adversarial Poisoning* and *NTGA*, RN-18 features actually *improve throughout training*, as we show in Figure 5.2 **(a)** and (**d**), despite low test accuracy during poison training (solid lines in Figure 5.2 **(a)** and **(d)**). On *AR* and ∘ *Random Noise*, test accuracy peaks early in training before dropping to random chance accuracy (solid lines in Figure 5.2 **(b)** and **(c)**). Still, when early checkpoints are used, one can use features from the poison-trained feature extractors to achieve high accuracy (dashed lines in Figure 5.2 **(b)** and

(c)). Accuracy curves for *Adversarial Poisoning* are unique because images are perturbed with error-maximizing noise, which correspond to actual features models use during classification [92]. In this case, DFR is reweighting useful, existing features for classification, leading to higher test accuracy. On the other hand, *Random Noise* and *AR* poisons do not perturb images with useful features; instead, both perturb with synthetic noise. In these cases, useful features are still learned during poison training, but only in the first epochs. As training progresses, the model checkpoints are continually corrupted by synthetic noise features which cannot be useful for classification despite reweighting.

As we show in Table 5.1, ∘ *Unlearnable Examples* are very effective at corrupting network weights during training; using DFR on checkpoints from poison training only yields a maximum test accuracy of 34.09%, nearly 2% worse than a randomly initialized feature extractor. Six of the eleven unlearnable datasets we consider yield checkpoints which achieve DFR test accuracies that are 10% or more higher than a randomly initialized feature extractor. Generally, it is interesting that unlearnable dataset methods with the lowest max DFR test accuracy are those which use error-minimizing perturbations in some capacity, *e.g.* *Unlearnable Examples*, *Robust Unlearnable Examples*, and *OPS+EM*. Analyzing test loss follows trends from test accuracy: *Adversarial Poisoning*, *AR*, *NTGA*, and *Random Noise* achieve lowest losses – and those poisoned checkpoints also have the highest DFR test accuracy in Table 5.1. More interestingly, we find that *all* poisoned models have a lower loss than the randomly initialized model.[2] This reinforces our claim that models learned useful features from poisoned data.

Overall, our results suggest that network weights during poison training are not fully corrupted in many cases. While poison-trained networks may be evaluated to have low test

[2]On 10 class classification task, the expected random chance loss is $-\ln(\frac{1}{10}) \approx 2.302$.

accuracy, we show that, for some unlearnable datasets, the networks have learned generalizable features which can be reweighted for high test performance. It is entirely possible that these features, which encode the original data amount to useful image information that the original data owner did not want incorporated into an ML system.

Privacy concerns are often cited as primary motivation for unlearnable datasets, yet our results raise concerns about this framing. A model trained on an apparently unlearnable dataset might have low test error, but this does not imply, as we show, that it does not contain usable information about the original data, and might not keep promises to protect data privacy.

5.2.1 Additional Model Architectures

We consider three more model architectures: VGG-16 [85], GoogLeNet [87], and ViT [60]. Our ViT uses a patch size of 4. For RN-18 and VGG-16, feature vectors are 512-dimensional. Feature vectors for GoogleNet are 1024-dimensional. The ViT class token is 384-dimensional.

In Table 5.2, we find that across architectures, *Adversarial Poisoning* data is easiest to extract generalizable features from. Surprisingly, ViT is most effective at learning generalizable features from *all* unlearnable datasets, achieving more than 7% test accuracy improvement over a randomly initialized ViT in all cases. For example, using only $5,000$ clean CIFAR-10 samples can be used to achieve nearly 69% test accuracy, while using the same clean samples can only achieve 37.40% test accuracy on a randomly initialized ViT. The GoogleNet architecture weights are seemingly more easily corrupted during training; Max DFR Test Accuracy for *Unlearnable Examples*, *AR*, *Robust Unlearnable*, and other datasets is much lower than test accuracy from a finetuned randomly initialized GoogleNet. Interestingly, the randomly initialized GoogleNet

Table 5.2: **Generalizable features can be learned from unlearnable datasets, using a variety of network architectures.** We report Max DFR Test Accuracy for each CIFAR-10 unlearnable dataset. In gray, we indicate test accuracy improvement/deterioration over DFR on the corresponding randomly initialized model architecture.

	Model Architecture		
CIFAR-10 Training Data	VGG-16	GoogleNet	ViT
None	35.69	48.08	37.40
Unlearnable Examples [28]	37.84 (+2.15)	41.08 (-7.00)	49.57 (+12.17)
Adversarial Poisoning [29]	64.73 (+29.04)	71.70 (+23.62)	68.97 (+31.57)
AR (ℓ_2) [32]	36.98 (+1.29)	40.12 (-7.96)	60.53 (+23.13)
NTGA [34]	56.03 (+20.34)	61.24 (+13.16)	60.53 (+23.13)
Robust Unlearnable [42]	39.13 (+3.44)	40.59 (-7.49)	49.10 (+11.70)
LSP [33]	40.86 (+5.17)	58.22 (+10.14)	50.95 (+13.55)
OPS+EM [38]	31.31 (-4.38)	38.57 (-9.51)	49.73 (+12.33)
◦ OPS [38]	39.63 (+3.94)	52.02 (+3.94)	56.04 (+18.64)
◦ Unlearnable Examples [28]	30.47 (-5.22)	36.32 (-11.76)	44.90 (+7.50)
◦ Regions-4 [44]	43.29 (+7.60)	48.65 (+0.57)	52.60 (+15.20)
◦ Random Noise	72.08 (+36.39)	62.19 (+14.11)	55.58 (+18.18)

feature extractor achieves the highest DFR test accuracy.

5.2.2 Additional Base Datasets

We consider three additional base datasets for four unlearnable dataset methods. We use an ImageNet [96] subset of the first 100 classes, following [28]. The train split consists of $129,395$ images, while the test split consists of $5,000$ images.

Our SVHN [90], CIFAR100 [39], and *Adversarial Poisoning* ImageNet subset datasets contain perturbations of size $\|\delta\|_\infty \leq \frac{8}{255}$. *Unlearnable Examples* ImageNet [96] subset contains perturbations of size $\|\delta\|_\infty \leq \frac{16}{255}$, following their open-source repository. We generate the *Adversarial Poisoning* [29] ImageNet subset from published source code using 1 PGD restart, as opposed to 8 due to computation time. Our SVHN and CIFAR-100 *Adversarial Poisoning*

Table 5.3: **Generalizable features can be learned from unlearnable datasets of different underlying distributions.** We report Max DFR Test Accuracy for each unlearnable dataset. RN-18 checkpoints are trained on SVHN, CIFAR-100, and ImageNet subset unlearnable datasets, and DFR is performed using $5,000$ clean samples from the corresponding base dataset.

	Finetune Data		
Training Data	SVHN	CIFAR-100	ImageNet
None	32.05	8.13	3.64
Unlearnable Examples [28]	26.76	16.12	8.44
Adversarial Poisoning [29]	87.06	44.37	20.22
∘ Unlearnable Examples [28]	22.48	10.32	7.88
∘ Random Noise	27.35	47.41	23.30

datasets use 3 PGD restarts. On clean SVHN, CIFAR-100, and ImageNet subset, RN-18 achieves 96.33%, 74.14%, 78.92% test accuracy respectively.

In Table 5.3, we again show that *Adversarial Poisoning* unlearnable data can be easily used to extract generalizable features regardless of the underlying distribution (base dataset of SVHN, CIFAR-100, or ImageNet). For SVHN, *Adversarial Poisoning* is the only dataset from which the trained feature extractor performs better in Max DFR Test Accuracy (87.06%) over a randomly initialized RN-18 (32.05%). Error-minimizing perturbations of *Unlearnable Examples* tend to be most effective at corrupting weights during training, regardless of underlying finetune data.

5.3 Linearly Separable Perturbations are Not Necessary

The work of Yu et al. [33] demonstrated that unlearnable datasets contain linearly separable perturbations and suggested that linear separability is a necessary property of unlearnable datasets. Motivated by this result, Wen et al. [99] create an unlearnable dataset with entangled features (EntF). However, separability is qualitatively evaluated through t-SNE visualizations, which is

Table 5.4: **Linearly separable perturbations are not necessary to create unlearnable datasets.** We train a linear logistic regression model on perturbations and report train accuracy. High train accuracy indicates linear separability of perturbations. Unlike perturbations from other unlearnable datasets, Autoregressive Perturbations (AR) are not linearly separable and are, in fact, less separable than clean CIFAR-10 images.

Training Data	Train Accuracy
Clean	53.88
Unlearnable Examples [28]	100
Adversarial Poisoning [29]	100
AR (ℓ_2) [32]	**39.58**
NTGA [34]	100
Robust Unlearnable [42]	100
LSP [33]	100
OPS+EM [38]	100
∘ OPS [38]	99.93
∘ Unlearnable Examples [28]	100
∘ Regions-4 [44]	100
∘ Random Noise	100

different from the linear separability test we perform here. More specifically, t-SNE cluster separability should not be equated to the linear separability we measure because it is possible to have linearly separable data that, when plotted using t-SNE, appears not separable. In other words, the EntF [99] unlearnable dataset could still contain linearly separable perturbations. In contrast, by finding a counterexample, we confirm that linear separability is not necessary to have an effective unlearnable dataset.

Following Yu et al. [33], we train linear logistic regression models on image perturbations by subtracting the clean image from the perturbed image. Given the publication of new unlearnable datasets, we add to Yu et al.'s analysis by including *AR*, *Robust Unlearnable Examples*, and *OPS* poisons. It is possible to obtain image perturbations given our access to clean CIFAR-10, but should not be possible in practice if the clean version of the unlearnable dataset is unavailable to the public. We optimize the logistic regression weights (3072×10 parameters for CIFAR-10)

using L-BFGS [98] for 500 steps using a learning rate of 0.5. For reference, we also train a logistic regression model on clean CIFAR-10 image pixels and report the train accuracy in the first row of Table 5.4. For results on the linear separability of the perturbed *images*, see Section 5.3.1.

While most current unlearnable datasets contain linearly separable perturbations, as was initially shown by Yu et al. [33], *AR* perturbations stand out as a counterexample in Table 5.4. After training on *AR* perturbations, a logistic regression model can only achieve 39.58% train accuracy while all other unlearnable datasets we tested contain perturbations that are almost perfectly linearly separable. Because most unlearnable dataset methods developed from diverse optimization objectives result in linearly separable perturbations, we posit that linear separability is an easy solution for optimization of Eq. 2.1 and 2.2 to find. For unlearnable datasets that are not optimized, class-wise perturbations are trivially linearly separable, so long as no two class perturbations are the same. Thus, although *AR* perturbations are not linearly separable, they remain separable by a simple 2-layer CNN [32], suggesting that simple perturbations may be a more accurate way of defining effective unlearnable perturbations.

Being the first counterexample of its kind, we believe there may be underexplored unlearnable dataset methods. This finding sheds light on the complexity of unlearnable datasets, whose behavior depends on choices of loss function, optimizer, and even network architecture. An important detail about *AR* perturbations is that they are not optimized – they are generated [32]. We leave to future work an investigation as to why most optimized perturbations in unlearnable datasets to date result in linear separability.

Table 5.5: **NTGA contains the least linearly separable images, while AR(ℓ_2) images are most comparable to the clean distribution. Other unlearnable datasets become *more* linearly separable.** We train a linear logistic regression model on poison images and report train accuracy. High train accuracy indicates linear separability of poison images. Mean and one standard deviation computed from 10 independent runs.

Training Data	Train Accuracy
Clean	53.94 ± 0.02
Unlearnable Examples [28]	100.00 ± 0.00
Adversarial Poisoning [29]	62.40 ± 0.01
AR (ℓ_2) [32]	53.97 ± 0.02
NTGA [34]	31.48 ± 0.02
Robust Unlearnable [42]	77.21 ± 0.01
LSP [33]	100.00 ± 0.00
OPS+EM [38]	100.00 ± 0.00
◦ OPS [38]	100.00 ± 0.00
◦ Unlearnable Examples [28]	100.00 ± 0.00
◦ Regions-4 [44]	100.00 ± 0.00
◦ Random Noise	100.00 ± 0.00

5.3.1 Evaluating Linear Separability of Poison Images

In the previous section, we document the linear separability of perturbations from various poisons, as in Yu et al. [33]. Poison images, on the other hand, behave slightly differently. In Table 5.5, we report logistic regression train accuracy on various CIFAR-10 poison images. We find that *Unlearnable Examples, LSP, OPS+EM*, and class-wise poisons have linearly separable poison images, but the remaining poisons we consider do not.

5.4 Learning from Unlearnable Datasets with Class-wise Perturbations

In this section, we develop a method to learn from unlearnable datasets. Learning the most predictive features in an unlearnable dataset can amount to learning the perturbation itself. We leverage this intuition in our Orthogonal Projection attack, which works by learning a linear

Algorithm 3 Orthogonal Projection

Input: Unlearnable Dataset, $(X, Y) \in \widetilde{\mathcal{D}}_{\text{train}}$
Output: Recovered Dataset, (X_r, Y)

while not converged **do**
    Sample batch (x, y) from $\widetilde{\mathcal{D}}_{\text{train}}$
    $W \leftarrow W - \eta \nabla_W L\big(W^T x, y\big)$
end while
Perform QR decomposition on W to obtain Q matrix
$X_r \leftarrow X - QQ^T X$

model, then projecting the data to be orthogonal to linear weights. In other words, we remove the most predictive dimensions of the data. For unlearnable datsets perturbed by class-wise, linearly separable perturbations, the most predictive dimensions are easily found by a linear model. Attacks for learning from class-wise perturbed unlearnable datasets have included using diffusion models [100], adversarial training [28], and error-maximizing augmentations [101]. In contrast to these techniques, our method is simpler and computationally cheaper.

5.4.1 Orthogonal Projection Method

Unlearnable datasets with class-wise perturbations are so simple that visualizing the average image of a class can expose the class perturbation (see Figure 5.4). In order to adaptively learn these perturbations and remove them from poisoned data, we propose a method that first learns the simple perturbations and then orthogonally projects samples to omit them. Our Orthogonal Projection method is designed to exploit class-wise perturbations by design and is meant to emphasize why class-wise perturbations should not be relied on for creating unlearnable datasets.

Given an unlearnable dataset, we train a simple logistic regression model on poison image pixels, in an attempt to optimize the most predictive linear image features. The result is a learned feature matrix W, on which we perform a QR decomposition such that $W = QR$,

where Q consists of orthonormal columns. Image samples X from the unlearnable dataset can be orthogonally projected using the matrix transformation $I - QQ^T$. Orthogonal projection ensures that the dot product of a row of X with every column of Q is zero, so that learned perturbations from the logistic regression model are not seen during the final training run on recovered data. The recovered image samples are then written as $X_r = (I - QQ^T)X$. Finally, we train the final network on recovered data, X_r. A detailed algorithm is shown in Algorithm 3.

The added computational cost of Orthogonal Projection over standard training is only in training the logistic regression model and projecting the dataset, which can be done once. For CIFAR-10, we train the logistic regression model for 20 epochs using SGD with an initial learning rate of 0.1, which decays by a factor of 10 on epochs 5 and 8. We find that training the linear model for longer is often detrimental, potentially because of overfitting.

Additional Intuition using CIFAR-10 as an example Assume CIFAR-10 images of shape $(3, 32, 32)$. Each column i of W (optimized in Alg 3, Lines 1-4) is a 3072-dimensional vector that represents the most predictive image feature for class i. This step serves as recovery of the perturbation. After the QR decomposition of W, Q consists of orthonormal columns that form a basis for the column space of W. When we say Orthogonal Projection "ensures that the dot product of a row of X with every column of Q is zero," (i.e., $X_r \cdot Q = 0$) this means that every recovered image vector does not contain any linearly separable component (i.e., does not contain any column of Q as a component). Alg. 3, Line 6 ensures image vectors and columns of Q are orthogonal and so the dot product is 0. The "recovered" data thus has 10 dimensions (approximations of the 10 perturbations) removed.

Table 5.6: **Subtracting the average class image from training images is not effective.** For *LSP* and *OPS*, datasets with class-wise perturbations, our Orthogonal Projection attack improves CIFAR-10 test accuracy over simply subtracting the average class image at training time.

	Attack	
Training Data	Class-Avg Subtract	Ortho Proj (Ours)
LSP [33]	13.05	87.99
∘ OPS [38]	12.62	87.94

Subtracting a Class-wise Image Given that the goal of Orthogonal Projection is to extract perturbations from poison images, it is reasonable to consider visualizing the average image of a class for class-wise poisons like *LSP* and *OPS*. In Figure 5.4, we see that class-wise average images somewhat reveal class-wise perturbations, but the results are not clear enough to be useful. In contrast, class-wise patterns are clearly present in learned weights from logistic regression.

For training models on class-wise perturbed data, one might consider subtracting the class-wise average image from each class. However, simply subtracting this average class image from each image does not remove the poisoning effect. Additionally, because we do not know the true class at inference time, we cannot subtract the class image, resulting in a distribution mismatch between train and test sets. This trivial method of subtracting average class images is compared to Orthogonal Projection in Table 5.6.

5.4.2 Orthogonal Projection Results

We compare our attack against ℓ_∞ adversarial training, as previous work has shown that adversarial training is a principled defense [35] against unlearnable datasets. For adversarial training, we use a 3-step PGD adversary with a perturbation radius of $\frac{8}{255}$ in ℓ_∞-norm. Note that large adversarial training perturbation radii harm clean accuracy [57]. We use SGD with

Table 5.7: **Orthogonal Projection can make class-wise unlearnable data learnable.** Especially for unlearnable datasets with class-wise, linearly separable perturbations, our Orthogonal Projection attack improves CIFAR-10 test accuracy over ℓ_∞ Adversarial Training at a fraction of the computational cost. We train RN-18 using different kinds of unlearnable training data and, for every attack, we indicate accuracy improvement/deterioration over standard training (no attack) in gray. For every dataset, we bold the highest test accuracy achieved.

	Attack		
Training Data	None	Adv Training	Ortho Proj (Ours)
Clean	94.37	87.16 (-7.21)	90.16 (-4.21)
Unlearnable Examples [28]	24.56	**85.64** (+61.08)	65.17 (+40.61)
Adversarial Poisoning [29]	7.96	**85.32** (+77.36)	14.74 (+6.78)
AR (ℓ_2) [32]	13.77	**84.09** (+70.32)	13.03 (-0.74)
NTGA [34]	40.78	**84.85** (+44.07)	82.21 (+41.43)
Robust Unlearnable [42]	26.86	**87.07** (+60.21)	25.83 (-1.03)
LSP [33]	25.75	86.18 (+60.43)	**87.99** (+62.24)
OPS+EM [38]	20.11	12.22 (-7.89)	**39.43** (+19.32)
∘ OPS [38]	15.35	11.77 (-3.58)	**87.94** (+72.59)
∘ Unlearnable Examples [28]	14.49	85.56 (+71.07)	**89.98** (+75.49)
∘ Regions-4 [44]	10.32	85.74 (+75.42)	**86.87** (+76.55)
∘ Random Noise	10.03	86.36 (+76.33)	**90.37** (+80.34)

momentum of 0.9 and an initial learning rate of 0.1, which decays by a factor of 10 on epoch 75, 90, and 100.

We find our method is *more effective than adversarial training* when learning from unlearnable datasets corrupted in a class-wise manner. In Table 5.7, our Orthogonal Projection attack leads to greater test accuracy on all class-wise poisons prefixed by ∘. By visualizing the learned weights from the linear model in Figure 5.3, we see that the class-wise, linearly separable perturbations are recovered. Projecting orthogonal to these vectors effectively removes those features from the unlearnable dataset, allowing training to proceed without predictive, but semantically meaningless features. It should not be surprising that Orthogonal Projection is also effective against *OPS* (ICLR 2023), given that all images of a class have the same pixel modified. In Figure 5.3, we visualize the linear model weights and show that it is able to pinpoint the modified class

pixel, whose influence is effectively removed after projection in our method. We visualize additional linear model weights for other datasets in Section 5.4.3. Orthogonal Projection is also surprisingly effective against *NTGA* (ICML 2021), such that an RN-18 model can achieve 82.21% test accuracy when trained on orthogonally projected *NTGA* images. For seven of the eleven unlearnable datasets we consider, Orthogonal Projection can achieve above 80% test accuracy without adversarial training. Unlike the linear model from Section 5.3 which is trained on only perturbations, the linear model that we train for Orthogonal Projection trains on perturbed images and the resulting data is no longer linearly separable. For this reason, Orthogonal Projection struggles against *Adversarial Poisoning*, *AR*, and *Unlearnable Examples*; the training distribution of the linear model is more complex than just perturbations. Given that *OPS+EM* is a combination of *OPS* and class-wise error-minimizing noise from *Unlearnable Examples*, both Orthogonal Projection and adversarial training are ineffective. For both *OPS* and *OPS+EM*, adversarial training achieves approximately 12% accuracy, a dismal result that is expected given the unbounded *OPS* perturbations. A limitation of our approach is that, by orthogonally projecting data, we effectively remove K dimensions from the data manifold, where K is the number of classes in a dataset. While this may not be a problem for high-resolution images with tens of thousands of dimensions, this detail could impact applicability for low-resolution datasets.

To evaluate the efficacy of Orthogonal Projection for transformer networks, in Table 5.8, we train ViT with patch size of 4 on CIFAR-10 unlearnable datasets using different attacks. Our Orthogonal Projection method is competitive with adversarial training for all class-wise perturbed unlearnable datasets and most sample-wise perturbed datasets. Orthogonal Projection is the best performing method for *OPS+EM* and *OPS* (ICLR 2023). Note that *OPS+EM* and *OPS* are most difficult for adversarial training.

Table 5.8: **Orthogonal Projection can make class-wise unlearnable data learnable for ViT.** Especially for unlearnable datasets with class-wise, linearly separable perturbations, our Orthogonal Projection attack is competitive with ℓ_∞ Adversarial Training at a fraction of the computational cost.

	Attack		
CIFAR-10 Training Data	None	Adv Training	Ortho Proj (Ours)
Clean	84.99	76.38	74.14
Unlearnable Examples [28]	25.39	75.44	60.15
Adversarial Poisoning [29]	31.33	75.15	41.49
AR (ℓ_2) [32]	17.13	75.12	35.11
NTGA [34]	32.67	71.95	66.19
Robust Unlearnable [42]	28.24	78.03	37.34
LSP [33]	29.40	75.45	74.77
OPS+EM [38]	20.73	11.79	51.94
∘ OPS [38]	21.58	10.17	72.80
∘ Unlearnable Examples [28]	12.19	76.35	76.01
∘ Regions-4 [44]	15.00	75.96	67.20
∘ Random Noise	29.66	76.23	73.05

To evaluate the efficacy of Orthogonal Projection for new datasets, we consider an additional dataset: CIFAR-100. We train a RN-18 on four unlearnable dataset methods. During the first step of Orthogonal Projection, we train the logistic regression model for 60 epochs using SGD with an initial learning rate of 0.1, which decays by a factor of 10 on epochs 30 and 45.

In Table 5.9, we find that Orthogonal Projection performs better on class-wise unlearnable data than sample-wise perturbed data, as expected. At approximately the cost of standard training (Attack: None), Orthogonal Projection achieves gains of more than 20% test accuracy for sample-wise perturbed data and more than 40% test accuracy for class-wise perturbed data.

The susceptibility of using class-wise perturbations to craft unlearnable datasets should be expected, given that every image of a class contains the same image feature that is perfectly predictive of the label. Class-wise unlearnable datasets contain simple features that are perfectly correlated with the label and the linear logistic regression model is able to optimize features

Table 5.9: **Orthogonal Projection is competitive on CIFAR-100 class-wise unlearnable data.**

	Attack		
CIFAR-100 Training Data	None	Adv Training	Ortho Proj (Ours)
Clean	74.14	59.23	26.78
Unlearnable Examples [28]	8.11	58.29	28.44
Adversarial Poisoning [29]	5.93	57.60	25.24
○ Unlearnable Examples [28]	1.72	60.31	41.83
○ Random Noise	1.30	58.83	51.23

resembling the original perturbations. By design, our orthogonal projection attack is most effective against class-wise perturbed unlearnable datasets. It serves as evidence that class-wise, linearly separable perturbations cannot be relied upon for protecting data.

5.4.3 Visualizing Logistic Regression Weights

We visualize additional linear model weights (from the first step of Orthogonal Projection) for sample-wise perturbed unlearnable datasets in Figure 5.5, and for class-wise perturbed unlearnable datasets in Figure 5.6. We find that for *Adversarial Poisoning*, *AR*, and *Robust Unlearnable* the linear model learns features comparable to those found when training on clean data. We posit that because the diversity of perturbations in these datasets is higher, the linear model struggles to find predictive features to project away. In contrast, for class-wise perturbed data, Figure 5.6, demonstrates that the linear model can recover features that resemble the original class-wise perturbation.

5.4.4 Samples from Unlearnable Datasets

We visualize samples from sample-wise perturbed CIFAR-10 unlearnable datasets in Figure 5.7, and from class-wise perturbed unlearnable datasets (prefixed by ∘ throughout results) in Figure 5.8. NTGA is omitted due to data ordering of the publicly available poison. We also visualize SVHN samples in Figure 5.9 and CIFAR-100 in Figure 5.10.

5.5 Conclusion

We design experiments to test prevailing hypotheses about unlearnable datasets, and our results have practical implications for their use. First, while all unlearnable datasets cause low test accuracy for trained models, we demonstrate that generalizable features can still be learned from data. In many cases, high test accuracy can be achieved from a feature extractor trained on unlearnable datasets. If data were modified for protection, an adversary may still be able to train a reasonable feature extractor, a harm for data thought to be secure. We advocate for our evaluation framework to be used for auditing whether new unlearnable datasets prevent useful features from being learned. Second, we find that the reason unlearnable datasets cause low test accuracy is not as simple as previously thought. AR perturbations serve as a counterexample to the hypothesis that unlearnable datasets contain linearly separable perturbations. Our finding that AR perturbations are not linearly separable suggests there could be underexplored methods for protecting data using imperceptible perturbations. Although unlearnable datasets need not have linearly separable perturbations, if they do have them applied in a class-wise manner, we present an Orthogonal Projection attack that can effectively remove them. While learning the perturbations from perturbed images is challenging, our results imply class-wise perturbations

cannot be relied upon for data protection. We can learn a significant amount from unlearnable datasets. DNNs can learn generalizable features from unlearnable datasets assumed to be protected, linear separability is not a necessary condition for preventing generalization from unlearnable datasets, and class-wise perturbations can be optimized against and effectively removed.

Every finding in this chapter concerns *bounded* perturbations, and one might reasonably respond by abandoning the imperceptibility constraint. If perturbations are allowed to be large and visible, perhaps it can no longer be projected away. Chapter 6 tests that response on the Convolution-based Unlearnable Dataset (CUDA), whose class-wise blurs are unbounded by design. We find that dropping imperceptibility does not restore protection either.

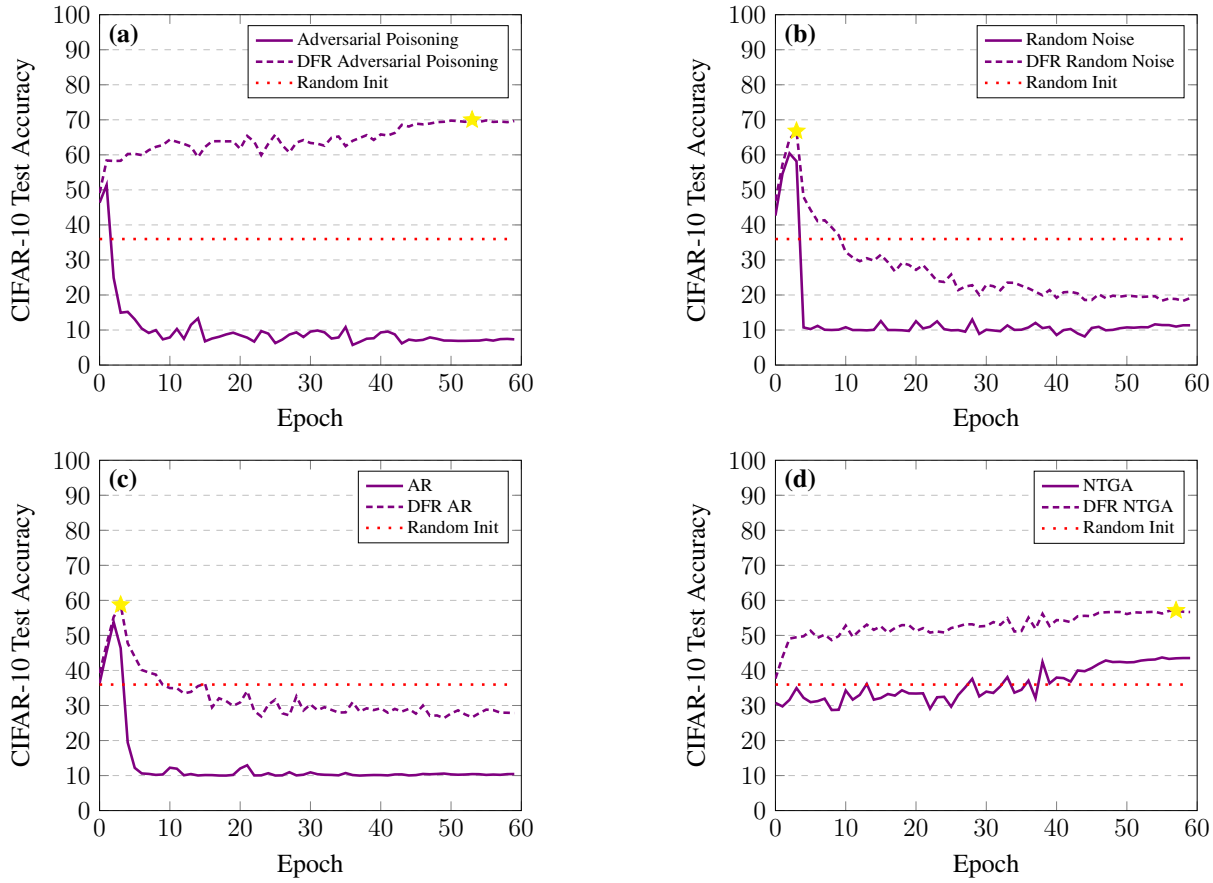


Figure 5.2: **Features learned during poison training can be reweighed for high test accuracy. (a)** When training on *Adversarial Poisoning*, RN-18 test accuracy (solid line) drops below random chance test accuracy. For the dashed line, each data point at epoch i represents the final test accuracy obtained after DFR using the poisoned checkpoint from epoch i of standard training. Using DFR, the final RN-18 poisoned checkpoint can achieve nearly 70% test accuracy (dashed line). Using DFR on a randomly initialized RN-18 only achieves 40% test accuracy (dotted line). **(b-d)** For different unlearnable datasets, at different points during poison training (marked by star), DFR test accuracy is more than 20% above the accuracy of a randomly initialized feature extractor, suggesting that features learned from unlearnable data are relevant for clean test set generalization.

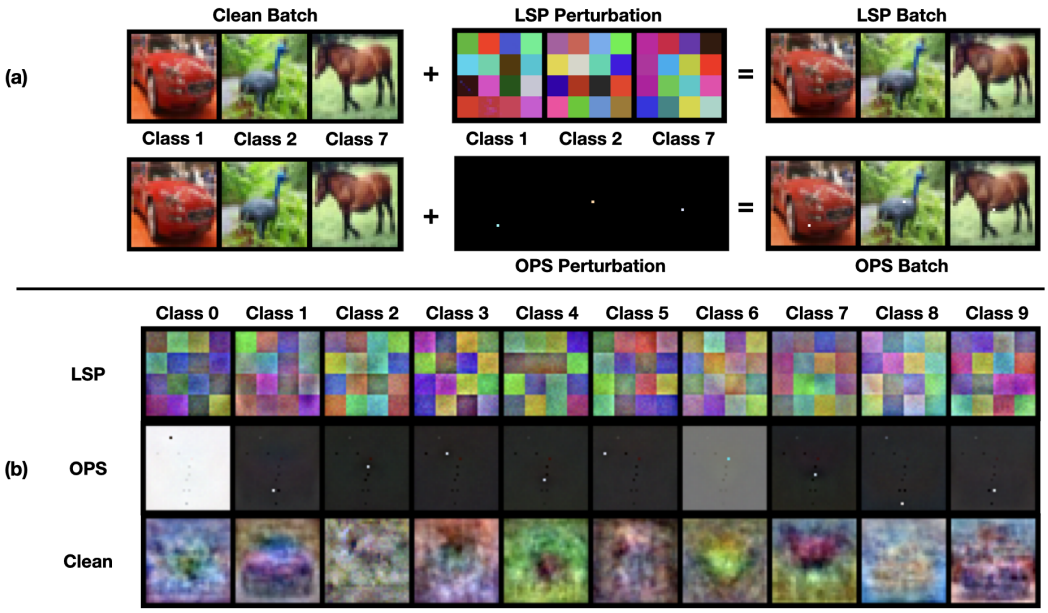


Figure 5.3: **Logistic regression learns linearly separable features.** In **(a)**, we visualize how LSP [33] and OPS [38] unlearnable data is created by taking clean image and adding a perturbation. In **(b)**, we visualize learned weights (W in Algorithm 3) after training a linear model on LSP and OPS images; we also include learned weights after training on clean CIFAR-10 for reference. Learned weights from LSP and OPS unlearnable data nearly match the original perturbation, while learned weights from clean data resemble the corresponding class. In our Orthogonal Projection attack, we project each perturbed image to be orthogonal to each of these learned weights (Algorithm 3, Line 6).

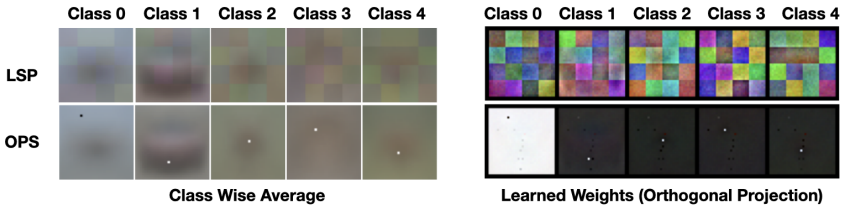


Figure 5.4: **Average class images display class-wise perturbations, but not as clearly as learned weights from logistic regression.** We compare the average image of a class **(Left)** and learned weights of a logistic regression classifier **(Right)** trained on image pixels (the first step of Orthogonal Projection method) to for *LSP* and *OPS* Poisons. While average image of a class does reveal the class-wise perturbation (block pattern for *LSP* and one highlighted pixel for *OPS*), the result is blurry and contains other semantic image features. Learned weights from Orthogonal Projection properly isolate the perturbation.

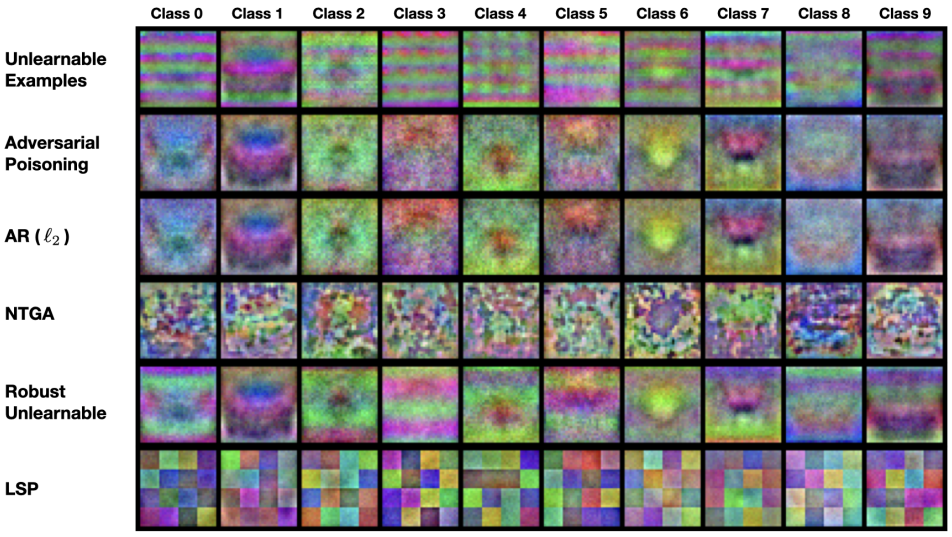


Figure 5.5: **Learned Weights from first step of Orthogonal Projection on sample-wise perturbed CIFAR-10 unlearnable datasets.** We visualize learned weights (W in Algorithm 3) after training a linear model on unlearnable datasets. Learned weights from *Adversarial Poisoning*, *AR*, and *Robust Unlearnable* resemble the learned weights from clean data (See Figure 5.3). In our Orthogonal Projection attack, we project each perturbed image to be orthogonal to each of these learned weights (Algorithm 3, Line 6).

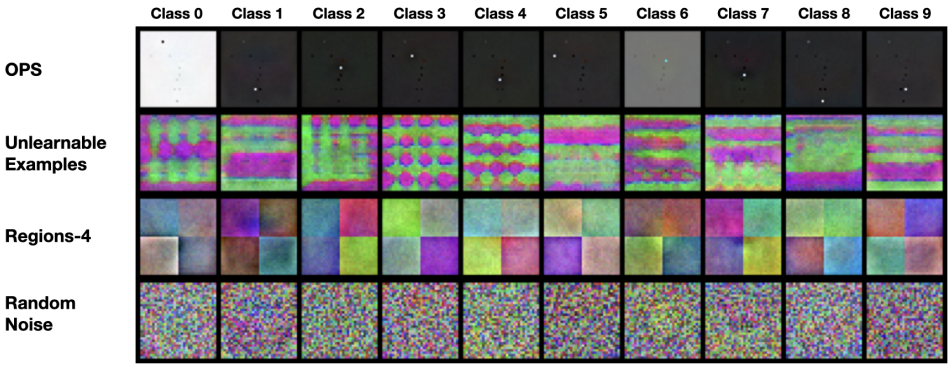


Figure 5.6: **Learned Weights from first step of Orthogonal Projection on class-wise perturbed CIFAR-10 unlearnable datasets.** We visualize learned weights (W in Algorithm 3) after training a linear model on unlearnable datasets. Learned weights appear to recover the added class-wise perturbation for all datasets. In our Orthogonal Projection attack, we project each perturbed image to be orthogonal to each of these learned weights (Algorithm 3, Line 6).



Figure 5.7: **Samples from CIFAR-10 unlearnable datasets.** We visualize the first 10 images from each sample-wise perturbed unlearnable dataset.

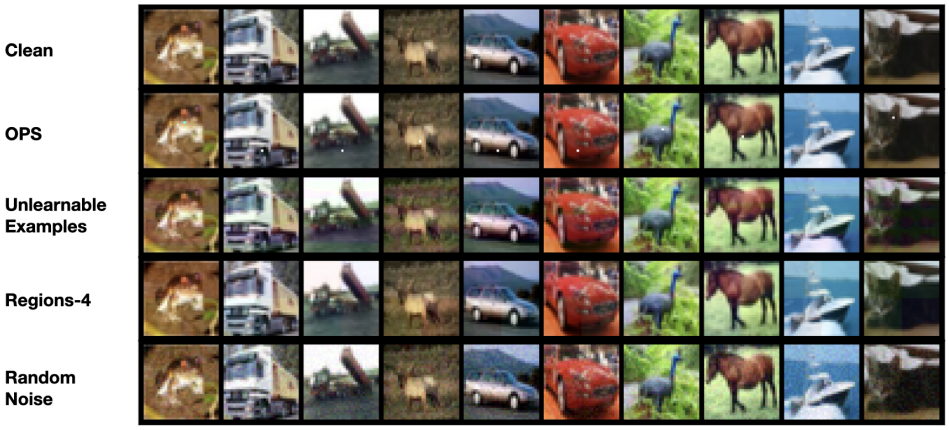


Figure 5.8: **Samples from CIFAR-10 unlearnable datasets.** We visualize the first 10 images from each class-wise perturbed unlearnable dataset.



Figure 5.9: **Samples from SVHN unlearnable datasets.**

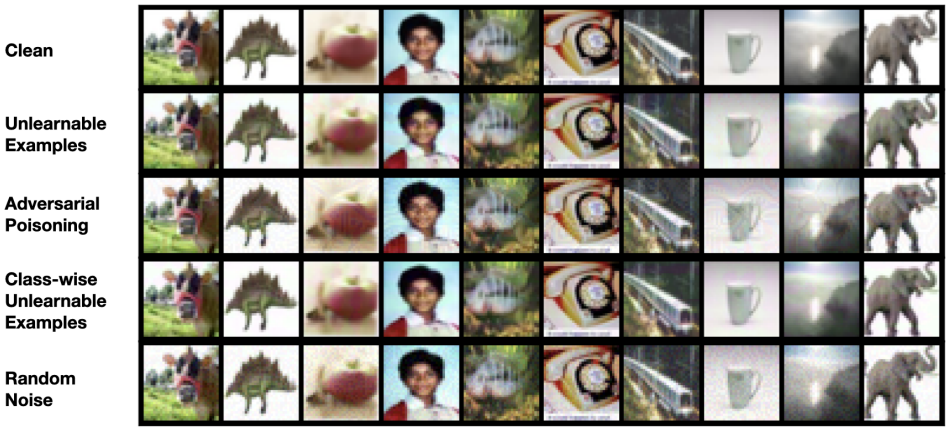


Figure 5.10: **Samples from CIFAR-100 unlearnable datasets.**

Chapter 6: Learning From Convolution-based Unlearnable Datasets

In Chapter 5 we showed that bounded, class-wise perturbations can be projected away. This chapter asks whether abandoning the imperceptibility constraint altogether provides the protection that bounded methods could not.

The construction of large datasets for deep learning has raised concerns regarding unauthorized use of online data, leading to increased interest in protecting data from third-parties who want to use it for training. The Convolution-based Unlearnable DAtaset (CUDA) method aims to make data unlearnable by applying class-wise blurs to every image in the dataset so that neural networks learn relations between blur kernels and labels, as opposed to informative features for classifying clean data. In this chapter, we evaluate whether CUDA data remains unlearnable after image sharpening and frequency filtering, finding that this combination of simple transforms improves the utility of CUDA data for training. In particular, we observe a substantial increase in test accuracy over adversarial training for models trained with CUDA unlearnable data from CIFAR-10, CIFAR-100, and ImageNet-100. In training models to high accuracy using unlearnable data, we underscore the need for ongoing refinement in data poisoning techniques to ensure data privacy. Our method opens new avenues for enhancing the robustness of unlearnable datasets by highlighting that simple methods such as sharpening and frequency filtering are capable of breaking convolution-based unlearnable datasets. [1]

[1]This chapter follows "Learning From Convolution-based Unlearnable Datasets" [102] by Dohyun Kim and

	Clean Data	CUDA Data	CUDA Data with AT	CUDA Data with RSK+FF (Ours)
CIFAR-10	94.34	23.25	44.40	**78.53**
CIFAR-100	74.41	17.33	34.34	**53.33**
ImageNet-100	81.76	2.68	38.68	**43.08**

Table 6.1: **Compared to common defenses like Adversarial Training (AT), our method applied to CUDA data can train a RN-18 to higher test accuracy on CIFAR-10, CIFAR-100, and ImageNet-100**. Our method uses Random Sharpening Kernels and Frequency Filtering (RSK+FF) to alter poisoned CUDA images during training. On CIFAR-10, CIFAR-100, and ImageNet-100, we achieve 55%, 36%, and 40% improvements, respectively, over training with CUDA data.

6.1 Introduction to Unbounded Unlearnable Datasets

To prevent the unauthorized scraping of data on the web, unlearnable datasets aim to render images unusable. Unlearnable dataset methods seek to find the best way to imperceptibly modify a training dataset so that a deep neural network (DNN) trained on the resulting data exhibits the largest generalization gap. The idea is that if data can be rendered useless, third parties are disincentivized from using the modified data.

Two types of unlearnable datasets exist: bounded and unbounded. Bounded methods [28, 29, 32, 42] attempt to make the perturbation as imperceptible to humans, while unbounded methods [36, 38] are more easily perceptible. Both types of unlearnable datasets face challenges. Existing works that explore *bounded* unlearnable methods have shown that Adversarial Training (AT) [28, 29, 34] and JPEG compression [103, 104] can increase the learnability of unlearnable data.

We propose a method for learning from *unbounded* unlearnable datasets like the Convolution-based Unlearnable DAtaset (CUDA). As bounded methods show weakness whether it be by AT

Pedro Sandoval-Segura, and was published in the AdvML-Frontiers Workshop at NeurIPS 2024.

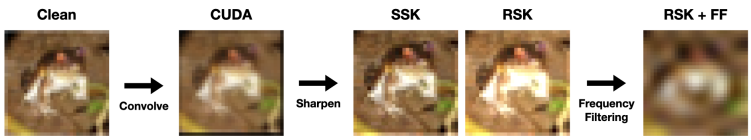


Figure 6.1: **Sharpening and Frequency Filtering of a CIFAR-10 Image**. We analyze the effect of standard and random sharpening kernels, both with a center value of 2.5. We find that the randomized sharpening kernel, denoted RSK, ensures images of the same class are sharpened differently. After sharpening, we decompose the image into spatial frequencies using DCT and filter out high frequencies (see Section 6.3.2).

or other transforms, Sadasivan et al. [36] argue that unbounded unlearnable methods are valid because 1) they do not obscure the semantics of the dataset, and 2) their class-wise blurring makes the model learn the relationship between the class-wise filters and their corresponding labels.

We demonstrate that it is possible to learn from CUDA datasets [36], despite their unbounded perturbations. We achieve a high test accuracy by sharpening the images from the CUDA dataset using our sharpening kernels followed by frequency filtering, achieving a 55%, 36%, and 40% improvement over CUDA standard training for CIFAR-10, CIFAR-100, and ImageNet-100, respectively. Our results suggest using random sharpening kernels should be a baseline against convolution-based perturbations.

6.2 Related Work

Unlearnable dataset methods make small perturbations to the whole training dataset so that DNNs trained on the modified data result in poor test accuracy [28, 29, 33]. Many types of perturbations have been explored: error-minimizing noise [28, 42], error-maximizing noise [29], autoregressive noise [32], low-frequency noise [44], and more. The modified datasets are typically constrained to "look like" the original dataset by requiring that the difference between

each original image and its modified image has an ℓ_p norm bounded by some ϵ. For example, a number of works bound perturbations using an ℓ_∞-norm [28, 29, 42], whereas other works use an ℓ_2 norm [32, 33]. This bound is also known as an "imperceptibility constraint," because it ensures that image modifications are not easily visible.

But there has been a growing movement allowing for unbounded perturbations [36, 38]. Unbounded perturbations are naturally more effective at inducing a generalization gap in DNNs due, in part, to the fact that perturbations are larger and the resulting images are noticeably modified. For example, the One-Pixel Shortcut (OPS) dataset [38] sets a single pixel that induces the largest gap from the mean pixel value of a class in a class-wise manner (all images of a class are modified in the same way). The Convolution-Based Unlearnable DAtaset (CUDA) dataset [36] also contains unbounded perturbations, where images are blurred class-wise. For example, on CIFAR-10, ten random blur kernels are generated and used to blur training images. This causes trained networks to learn relations between blur kernel and labels, *i.e.* a learning shortcut [1, 33] which is not informative at test-time. Nevertheless, Wu et al. [38] and Sadasivan et al. [36] argue that as long as the semantic information in an image is preserved, unbounded perturbations should be explored. After all, both works demonstrate that unbounded perturbations are more resistant to adversarial training and augmentations. CUDA is also effective against randomized smoothing [58], as well as Deconvolution-based adversarial training. This observation aligns with the data in Table 6.2, which shows that the ℓ_p norms of CUDA and OPS are substantially higher than their counterparts. Notably, CUDA exhibits a four to six-fold increase in its ℓ_2 norm compared to alternative methods.

The unbounded nature of CUDA, which causes a visible blur (see Figure 6.2), naturally makes it harder for models to learn, as the data is further away from the original training distribution.

	CUDA	EM [28]	TAP [29]	AR [32]	REM [42]	OPS [38]	R4 [44]
ℓ_2	**6.63**	1.49	1.45	0.99	1.32	1.07	<u>1.72</u>
ℓ_∞	<u>0.54</u>	0.03	0.03	0.09	0.03	**0.68**	0.03

Table 6.2: **Size of an average perturbation from different unlearnable datasets.** For each training image, we compute the norm of the difference between the modified image and the clean image. We average over all training images in CIFAR-10. CUDA perturbations have an ℓ_2-norm that is at least $3.8\times$ larger than other datasets. OPS is the only other dataset containing unbounded perturbations. We **bold** the largest norm and <u>underline</u> the second-largest norm.

Because it remains unknown whether recovering the blur kernels from CUDA training data is feasible, CUDA could appear like a reasonable approach to data security; a data owner could use the CUDA method to protect their data from being trained on. However, in this chapter, we demonstrate that random sharpening kernels and frequency filtering can make CUDA data learnable again.

While some theory can explain why CUDA is effective on data sampled from a Gaussian mixture model [36], our findings on standard datasets like CIFAR-10, CIFAR-100, and ImageNet-100 (a subset of the first 100 classes) demonstrate that classifiers trained on CUDA data can actually still generalize. There have been previous works that attempt to counter unlearnable datasets. Notably, Image Shortcut Squeezing (ISS) [103] uses Grayscale and JPEG transforms to counter low-frequency and high-frequency perturbations, respectively. However, the use of JPEG is hard to reason about because of the way JPEG removes information. The JPEG algorithm divides an image into 8x8 blocks then removes the high frequency information from each one independently. This means that it not only removes high frequencies, but the high frequencies from low frequency patches of an image - resulting in the removal of low frequency information as well. In addition to this, the precalculated quantization matrices for the chrominance and luminance channels are specifically designed to keep the image unchanged to the human eye.



Figure 6.2: **Comparison of the same image from different unlearnable datasets.** OPS [38] perturbs by adding noise to a singular pixel. AR [32] generates perturbations using a sliding window approach. R4 [44] is a grid-like additive perturbation. CUDA and OPS, being unbounded methods, exhibit a noticeable perturbation, compared to AR and R4 which have perturbations bounded by an ℓ_p-norm.

Thus, while JPEG compression may remove the harmful perturbation, it is difficult to pinpoint where exactly in the frequency spectrum this occurred. Prior to our work, it was also unknown whether JPEG compression could break[2] CUDA, a fundamentally different type of poison with relatively larger perturbations. In contrast, our method includes a decomposition of an image into spatial frequencies, allowing us to determine where perturbations lie.

6.3 Motivation

DNNs fail to generalize to the data because the perturbations are tied with the class labels [36, 38]. This phenomena is known as shortcut learning, where DNNs learn the simplest feature that correlates well with labels. In this section, we empirically verify that DNNs in fact do learn shortcuts when trained on unlearnable data.

6.3.1 Shortcut Learning

To isolate the effect of perturbations, we must ensure the model does not learn meaningful features from the training dataset. All the unlearnable methods we analyze employ a class-wise poison, creating a tie between perturbation and class label. Suppose K is the number of classes

[2]Because the purpose of an unlearnable dataset is to make data useless for training, to "break" an unlearnable dataset means to make the data useful again – to train a model to high test accuracy using the corrupted data.

Test Image Perturbation	Poisoned Training Data			
	CUDA [36]	OPS [38]	AR [32]	R4 [44]
$x_i + \delta_{y_i}$	93.17	86.33	100	99.99
$x_i + \delta_{y_{i+1}}$	6.38	0.86	0	0.25
$x_i + \delta_{\mathcal{U}(0,K)}$	14.88	12.41	10.40	10.42

Table 6.3: **Unlearnable datasets train models to learn perturbations.** Using poisoned models from four unlearnable datasets, we evaluate classification accuracy on a *modified test set*. Each of these unlearnable datasets applies perturbations in a class-wise manner: δ_{y_i} refers to a perturbation for class i (or generated by the i^{th} process in the case of AR Poisoning). The test dataset is modified in three ways: $x_i + \delta_{y_i}$ refers to the perturbation matching the class of the test image, $x_i + \delta_{y_{i+1}}$ refers to the perturbation not matching the test image, and $x_i + \delta_{\mathcal{U}(0,K)}$ refers to the perturbation being random. Achieving high accuracy on images not seen during training only when the right perturbation is present suggests that poisoned RN-18's learn a class perturbation, as opposed to any useful features from data.

for a given dataset, with $\delta \in \Delta_C$ being a perturbation. We denote the test image and test label as x_i and y_i respectively. We then apply δ to x_i in three different ways: perturbation matching test label, perturbation not matching test label, and perturbation being random. We test the poisoned model on these three sets of the test data and expect test accuracies of 100%, 0%, and $1/K\%$, respectively. If the model learned a shortcut, the perturbation's class should determine the predicted label. We expect 100% when the perturbation matches the test label because the perturbation's class is always the correct class label. We expect 0% when the test label does not match the perturbation, because the perturbation's class is always incorrect. We expect $1/K\%$ with K classes when the perturbation is uniformly random, because the perturbation will only match the correct label $1/K\%$ of the time on average. Table 6.3 demonstrates that poisoned models indeed learn shortcuts where the presence of a perturbation determines the model predictions.

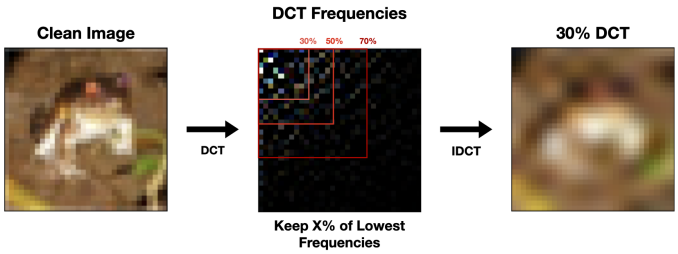


Figure 6.3: **DCT can be used to remove exact frequency bands.** DCT converts an image into a spatial frequency representation of the same size. The coefficients represent increasing frequencies from top-left (lowest) to bottom-right (highest). We retain the lowest $X\%$ of frequency coefficients by masking out the remaining higher frequencies. Then, we apply the inverse DCT (IDCT) to transform the modified frequency representation back into an image.

6.3.2 Frequency Filtering via Discrete Cosine Transform

Liu et al. [103] demonstrate that JPEG compression effectively counters high-frequency perturbations, while grayscale conversion mitigates low-frequency perturbations. Although their empirical results support the efficacy of these methods, they do not delve into further analysis of how JPEG compression alters the frequencies of poisoned images. The complexity of the JPEG algorithm, which involves six discrete steps, makes it challenging to pinpoint which aspects contribute to the improvements in robustness against adversarial attacks.

Instead, we use a frequency filtering (FF) method that uses the Discrete Cosine Transform (DCT), a component of JPEG compression, to systematically remove high frequencies from images. Figure 6.3 visualizes our method. It involves applying DCT to input images, keeping increasing percentages (10%, 20%, 30%...) of the DCT coefficients, and then performing an inverse DCT to reconstruct the images. This process generates a series of images predominantly composed of lower frequency information, with higher frequency components gradually introduced as a larger fraction of coefficients is retained, resulting in images that go from blurry to sharp (refer to Figure 6.5).

6.4 Method and Experiments

We identified shortcut learning behavior using unlearnable datasets like OPS, AR, and Regions-4. Now we discuss how we arrived at our method of learning from unlearnable datasets, and finally verify empirically that our method breaks said shortcuts.

6.4.1 Settings

Datasets and Models. For our experiments, we train ResNet-18 [76], and three datasets: CIFAR-10 [39], CIFAR-100 [39], and a 100-class subset of ImageNet [96], which we denote ImageNet-100. ImageNet-100 contains 129,395 train samples and 5,000 test samples. We create CUDA datasets for CIFAR-10, CIFAR-100, and ImageNet-100 using default settings from the original work: For CIFAR-10 and CIFAR-100, we use blur parameter $p_b = 0.3$ and kernel size of 3. For ImageNet-100, we use $p_b = 0.06$ and kernel size of 9. The percentage of poisoned data is 1.0, *i.e.* the whole dataset.

Training Hyperparameters. All classifiers are trained using cross-entropy loss. For both the CIFAR-10 and CIFAR-100 datasets, we train models for 60 epochs using SGD with a learning rate of 0.1, momentum of 0.9, and weight decay of 5×10^{-4}. We use a cosine annealing learning rate scheduler. We use a batch size of 512 for both, and normalize using the mean and standard deviation from each dataset. For data augmentations, we use random crops and random horizontal flips for CIFAR-10. For CIFAR-100, we additionally apply random rotations.

For ImageNet-100, we train ResNet-18 for 100 epochs using SGD with momentum of 0.9, and weight decay of 5×10^{-5}. We use a cyclic learning rate schedule which starts at 0.25, peaks

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -0.375 & 0 \\ -0.375 & 2.5 & -0.375 \\ 0 & -0.375 & 0 \end{bmatrix} \quad \begin{bmatrix} 0 & -\mathcal{N}(0.375, 0.1) & 0 \\ -\mathcal{N}(0.375, 0.1) & -\mathcal{N}(2.5, 0.1) & -\mathcal{N}(0.375, 0.1) \\ 0 & -\mathcal{N}(0.375, 0.1) & 0 \end{bmatrix}$$

Figure 6.4: **Left:** A standard sharpening kernel, using a center value of 5. **Middle:** A softer sharpening kernel with a center value of 2.5. **Right:** A random sharpening kernel where values are sampled independently from a normal distribution.

to 0.5 at epoch 2, and decays to 0 by epoch 100 (*i.e.* one cycle). For data augmentations, we use random resized crops and horizontal flips as data augmentations.

6.4.2 Random Sharpening Kernels + Frequency Filtering

Sharpening Kernels. In computer vision, sharpening kernels enhance image edges by emphasizing differences in adjacent pixel values. Figure 6.4 (left) shows a standard sharpening kernel. We find that a softer kernel with a center value of 2.5 (Figure 6.4 middle) outperforms the standard kernel with a center of 5. Applying this softer kernel to CUDA CIFAR-10 training data improves RN-18 test accuracy from 23.25% to 26.72%, a 3.47% increase. Despite the CUDA training data being more legible with the help of sharpening kernels, they alone cannot break the learned relationships between class-wise blur filters from CUDA and ground truth labels. Thus, we resort to randomized sharpening kernels.

The Random Sharpening Kernel (RSK). We propose to use random sharpening kernels (RSK) where we sample the center value from a normal distribution, then sample the neighboring values using the mean from the sampled center value (see Figure 6.4 right). RSK samples its center value from a normal distribution with a mean of 2.5. From the previous section with SSK, we have a test accuracy of 26.72% on CIFAR-10. After applying the RSK to the CUDA training data, we observe a 3.22% improvement to 29.94% test accuracy on CIFAR-10. In Table 6.4, we also include data for CIFAR-100 and ImageNet-100. An example of an RSK is in Figure 6.4 (c).

Frequency Filtering via DCT. Along with RSK, we consider FF (refer to Section 6.3.2) as a transform. We test this method on CUDA, and CUDA preprocessed by SSK and RSK. Our results are shown in Table 6.3. We observe that keeping the lowest 30 to 40% of frequencies is best for CIFAR-10 and CIFAR-100, while a smaller percentage of 5% is best for ImageNet-100. This difference is due to ImageNet images being of larger size (224×224). However, note that 5% of an ImageNet image's spatial frequencies is equivalent to 34% of the spatial frequencies in a CIFAR image. [3]

Putting it all together. As seen in Table 6.4, FF and RSK do not perform well by themselves against CUDA, but they excel when used together. After applying RSK with FF (30), we observe a test accuracy jump from 29.94% to 78.53% on CIFAR-10, an improvement of 48.59% over relying on RSK alone. We find that using an RSK with center 2.5 and FF with 30% of lowest frequencies kept is the best way to learn from CUDA.

	CIFAR-10	CIFAR-100	IMAGENET-100
CLEAN	94.34	74.41	81.76
CUDA	23.25	17.33	2.68
CUDA + SSK	26.48	23.82	5.04
CUDA + RSK	29.01	27.27	6.34
CUDA + FF	37.48 (10)	21.06 (30)	40.84 (5)
CUDA + SSK + FF	72.60 (30)	50.87 (30)	42.58 (5)
CUDA + RSK + FF	**78.53 (30)**	**53.33 (40)**	**43.08 (5)**

Table 6.4: **A breakdown of the transforms we use against CUDA.** It shows the test accuracies of the different combinations of transforms we use (refer to Section 6.4.2 for a breakdown). The parentheses next to results that use FF refers to the % of lowest frequencies kept. See Appendix 6.4.3 for an analysis on different percentages kept.

[3] The lowest 5% of spatial frequencies in an ImageNet image is represented by a 11×11 DCT coefficients matrix, which is $\frac{11}{32} \approx 34$ percent of a CIFAR image's spatial frequencies.

6.4.3 Analysis on Frequency Filtering

In this section, we analyze how datasets and different combinations of transforms behave under multiple percentages of kept DCT frequencies.

	Kept DCT Frequency Percentage									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Clean	40.74	37.48	46.55	31.73	44.85	91.40	92.16	92.57	93.18	**93.66**
CUDA	**37.48**	36.83	21.09	20.36	20.20	17.51	20.35	21.37	21.95	21.04
SSK	45.26	71.06	**72.60**	58.09	49.22	45.07	41.61	38.12	30.19	26.48
RSK	46.55	73.72	**78.64**	68.10	60.35	53.86	48.60	41.23	35.95	29.01

Table 6.5: **RSK is better than SSK.** This table displays test accuracies of ResNet-18 trained on CIFAR-10. CUDA refers to only CUDA convolved images. RSK refers to CUDA images convolved with RSK, and SSK refers to CUDA images convolved with SSK. It is observed that RSK has better performance than SSK on all frequency bands.

The baseline, clean CIFAR-10, performs as expected. As we allow the model to train on an increasing amount of higher frequency bands to the model, the test accuracy improves as high frequency components are useful for classification [105]. For the next three transforms, as seen in Table 6.5, we can see a trend: they perform best when most of the high frequencies are masked out; in other words, mostly blurry. This is interesting because from previous works, we know that high frequency components help DNNs generalize better. However, Table 6.5 shows that high frequency components can in fact "distract" the model from learning. We posit that sharpening kernels bring back crucial information needed for DNNs to generalize, and removing high frequencies works as an effective countermeasure to break the relation between the class-wise filters from CUDA and their corresponding labels.

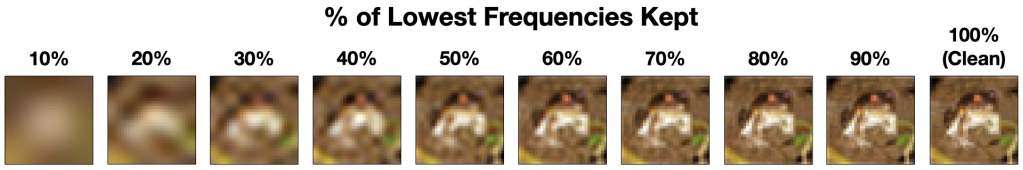


Figure 6.5: **Image reconstructions using different percentages of frequencies kept.** On the very left, we retain the lowest 10% of DCT frequencies. We progressively preserve higher frequencies, resulting in a series of blurry to clear images. These images are constructed using an image from the clean dataset.

	Kept DCT Frequency Percentage									
	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Clean	17.14	39.53	54.16	61.05	65.74	69.09	70.14	71.19	72.29	**72.80**
CUDA	13.50	21.04	**21.06**	13.76	12.74	11.03	11.47	13.00	12.65	15.35
SSK	19.10	40.54	**50.87**	50.57	47.12	44.30	39.88	29.85	24.44	23.82
RSK	19.90	42.47	52.58	**53.33**	52.84	49.55	46.08	35.44	29.23	27.27

Table 6.6: **Test accuracies of ResNet-18 trained on CIFAR-100.** We test our sharpening kernels, SSK and RSK, with DCT filtering (FF) on different percentages of kept DCT frequencies. We find that percentages of 30% to 40% work the best, similar to CIFAR-10.

	Kept DCT Frequency Percentage										
	5%	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
CUDA	**40.84**	38.76	4.14	2.86	3.28	2.02	2.76	2.40	2.88	2.96	2.74
SSK	**42.58**	39.04	4.46	5.96	5.42	7.08	5.90	4.72	4.62	5.98	5.38
RSK	**43.08**	39.64	8.76	7.32	5.62	7.44	6.26	6.28	8.54	6.50	7.52

Table 6.7: **Test accuracies of ResNet-18 trained on ImageNet-100.** We test our sharpening kernels, SSK and RSK, with DCT filtering (FF) on different percentages of kept DCT frequencies. We find that a percentage of 5% works the best. The lowest 5% of spatial frequencies in ImageNet images is represented by an 11×11 DCT coefficient matrix, which is $\frac{11}{32} \approx 34$ percent of a CIFAR image's spatial frequencies - it matches up with the percentages where highest accuracies were achieved with CIFAR-10 and CIFAR-100 (30% and 40% each).

6.5 Conclusion

As protection against data scraping and copyright infringement, unlearnable datasets must be robust against adversarial training and other attacks. We identified that shortcut learning is

	CUDA[36]	OPS[38]	AR[32]	R4[44]
Baseline	23.25	31.05	18.41	14.04
RSK + FF	78.53 (30)	87.03 (40)	67.23 (20)	23.17 (10)

Table 6.8: **Test accuracies of ResNet-18 trained on different sets of unlearnable datasets on CIFAR-10.** We tested various unlearnable datasets against our best method, RSK + FF. We find that it performs even better on OPS than CUDA, but performs poorly on Autoregressive Poisons and Regions-4. The parentheses notes the lowest kept percentage of DCT frequencies, and we reported the highest accuracy after testing from 10% to 100%.

a technique used by unlearnable datasets to fool DNNs. Based on the fact that CUDA is a class-wise blur, we designed a random sharpening kernel that breaks the relationship between the filters and their corresponding labels. We also used the intuition of JPEG compression being capable of removing perturbative information, and designed a method to filter out specific frequencies using the Discrete Cosine Transform. We found that we were able to boost test accuracy from CUDA-convolved CIFAR-10, CIFAR-100, and ImageNet-100 by 55.3%, 36%, and 40.4%, respectively. From our results, we find that unlearnable datasets can be exploited by simple image transformations, making them not as unlearnable as once thought.

The evidence in Chapters 5 to 6 supports a more general claim about data protection: whether protective perturbations are bounded or unbounded, additive or convolutional, sample-wise or class-wise, they must remain simple enough for a network to prefer and preserve the image's semantics. Yet these two requirements together leave enough structure for a cheap transform to exploit. Chapter 7 asks whether this claim extends beyond discriminative training to generative editing, where protection applies to a single image rather than a dataset.

Chapter 7: JPEG Compressed Images Can Bypass Protections Against AI Editing

Recently developed text-to-image diffusion models make it easy to edit or create high-quality images. Their ease of use has raised concerns about the potential for malicious editing or deepfake creation. Imperceptible perturbations have been proposed as a means of protecting images from malicious editing by preventing diffusion models from generating realistic images. However, we find that the aforementioned perturbations are not robust to JPEG compression, which poses a major weakness because of the common usage and availability of JPEG. We discuss the importance of robustness for additive imperceptible perturbations and encourage alternative approaches to protect images against editing. [1]

The results in this chapter are motivated by the brittleness of imperceptible perturbations to JPEG compression [104]. While Chapter 5 explored settings where unlearnable datasets are learnable, the work of Liu et al. [103] demonstrates that JPEG compression is yet another strong counter attack, making data highly learnable by neutralizing perturbations.

7.1 Protection Methods

Given a user-defined text description of an image, diffusion models [107, 108] excel at generating high-quality image samples. Diffusion models can also be used to edit specific parts

[1] This chapter follows the preprint "JPEG Compressed Images Can Bypass Protections Against AI Editing" [106] by Pedro Sandoval-Segura, Jonas Geiping, and Tom Goldstein.

of an existing image, guided by a text-prompt [108]. Previously, photo editing software and expertise was required to make serious scene adjustments or object additions, but now it can be done with a simple natural language phrase. The barrier to creating new images is relatively low, opening the door to malicious actors who could edit images or create deepfakes with a single click. To prevent the malicious editing of images, recent work has proposed using imperceptible perturbations to protect images from being modified [109, 110]. As with other methods which modify images to prevent a deep learning model from accomplishing a task, releasing protected images online exposes the owner to a static defense vs. adaptive attack scenario [95]. In this setting, the protected images are required to withstand adaptive attacks from current methods and all future methods, a nearly impossible requirement. By successfully editing protected images, we demonstrate that JPEG compression remains a necessary baseline for protection methods that use imperceptible noise. The widespread use of JPEG transformations [111], which may occur inadvertently (*e.g.* , if JPEG is the default file format for capturing screenshots), suggests that optimized perturbations for image protection are exceedingly fragile. The robustness of perturbations is especially important in situations where image misuse by adversaries is of concern. All experiments are conducted using the open-source notebooks from the Photoguard repository [109], which employs the Stable Diffusion Model (SDM) v1.5 [108].

7.1.1 Photoguard: Protecting Against Malicious AI-Powered Image Editing

Salman et al. [109] propose methods for protecting images against manipulation by Stable Diffusion [108]. Their software, *Photoguard*, protects images against manipulation by adding an imperceptible adversarial perturbation designed to force a target diffusion model to generate

unrealistic images. The authors propose two different protection techniques: an encoder attack and a diffusion attack.

Encoder attack. The goal is to find an imperceptible perturbation δ_{encoder} that, when added to the original image, results in a protected image that the SDM considers similar to a gray target image:

$$\delta_{\text{encoder}} = \underset{\|\delta\|_\infty \leq \epsilon}{\arg\min} \|E(x+\delta) - z_{\text{target}}\|_2^2 \tag{7.1}$$

where x is the image to be protected, z_{target} is some target latent representation, and E is the image encoder. For a $512 \times 512 \times 3$ dimensional image, $\epsilon = 0.1$ and 40 steps of PGD with step-size $\frac{0.1}{40} \times 6$ is used to solve the optimization.

Diffusion attack. The goal is to find an imperceptible perturbation $\delta_{\text{diffusion}}$ that, when added to the original image, results in a protected image that targets the diffusion process itself.

$$\delta_{\text{diffusion}} = \underset{\|\delta\|_\infty \leq \epsilon}{\arg\min} \|f(x+\delta) - x_{\text{target}}\|_2^2 \tag{7.2}$$

where f is the SDM, x is the image to be protected, x_{target} is the target image to be generated. Eq. (7.2) is solved approximately using PGD, backpropagating through only 4 diffusion steps due to GPU memory requirements [109].

Salman et al. [109] note that Photoguard's effectiveness may diminish when protected images undergo transformations, and propose methods for making the imperceptible perturbation more robust [112]. Protecting images using an imperceptible perturbation optimized using either of the attacks of Sec. 7.1.1 is a interesting and novel contribution, but our research has found

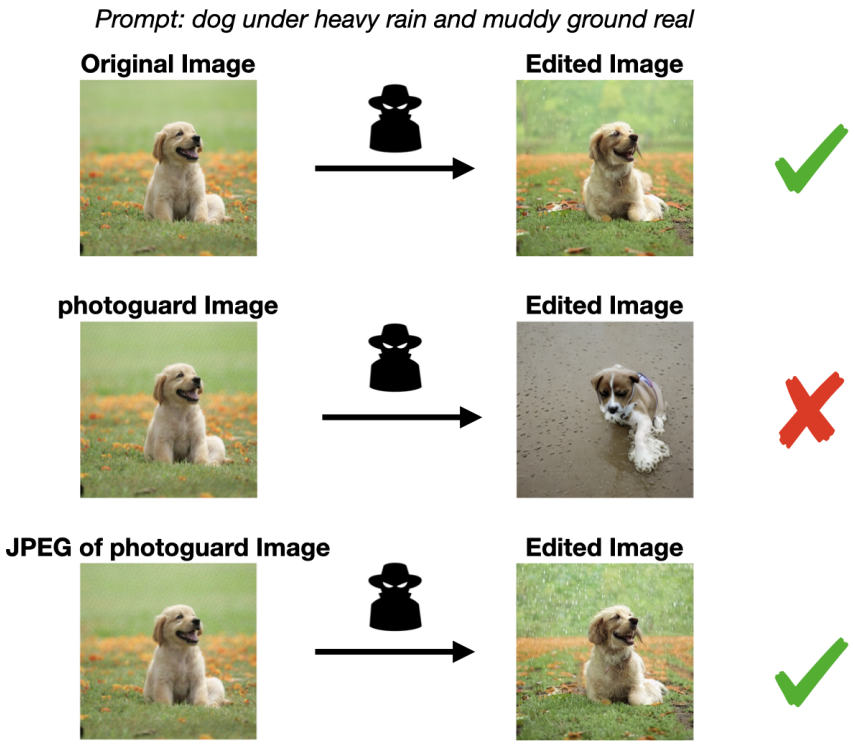


Figure 7.1: **JPEG compression allows an adversary to modify a protected image found online.** First row: Given a text prompt, an adversary can make desired edits to an input image using a diffusion model. Second row: photoguard (Encoder attack) [109] protects the original image before an adversary can access it by adding an imperceptible perturbation. When the adversary edits the photoguard image, they are unable to maintain the original subject. Third row: By JPEG compressing the photoguard image, an adversary can edit the photoguard image while maintaining the original subject and adding key visual features of the text prompt.

it is more brittle than expected to JPEG distortion, which poses a major weakness due to the common usage and availability of JPEG. Previous work has shown that a differentiable JPEG module can be added to the loss function to create JPEG-resistant adversarial examples [113]. One can imagine making a similar, simple modification to Eq. (7.1) and Eq. (7.2) so that the perturbations are JPEG-resistant. But the question remains: how many times does one want to continue the back and forth between the data protector and adversary?

7.1.2 Glaze: Protecting Against Style Mimicry

Shan et al. [110] propose a method to prevent fine-tuned diffusion models from learning a particular artist's style. In essence, imperceptible perturbations are optimized so that SDM finetuning on protected artist images, in the manner described by DreamBooth [114], is unsuccessful. In this way, new images generated by the finetuned model are unable to replicate the style captured in the protected subset of artist images. The proposed solution is unique and the authors perform experiments to test the robustness of their perturbations to Gaussian noise and JPEG image compression. However, only three compression levels between 20 and 10 are considered. Our experiments suggest that considering more subtle compression levels (*e.g.* , between compression levels 95 and 65) are necessary. Excessive compression, even without the use of optimized imperceptible perturbations, could potentially damage the image to the extent that the style appears to not have been replicated.

Evaluating whether generated images replicate a particular style, or whether generated content satisfies some other criteria in general, presents a challenge due to its subjective nature. Our experiments suggest that imperceptible perturbations must possess greater robustness against a wider range of image transformations than those commonly considered, as final transformations could be made by a malicious actor.

7.2 Bypassing Photoguard Protection

At a high level, Photoguard adds imperceptible noise to an image so that the image encoder in a diffusion model is unable to encode the image reasonably, resulting in downstream error by the denoising network. The idea being that if an adversary can only produce unrealistic images,

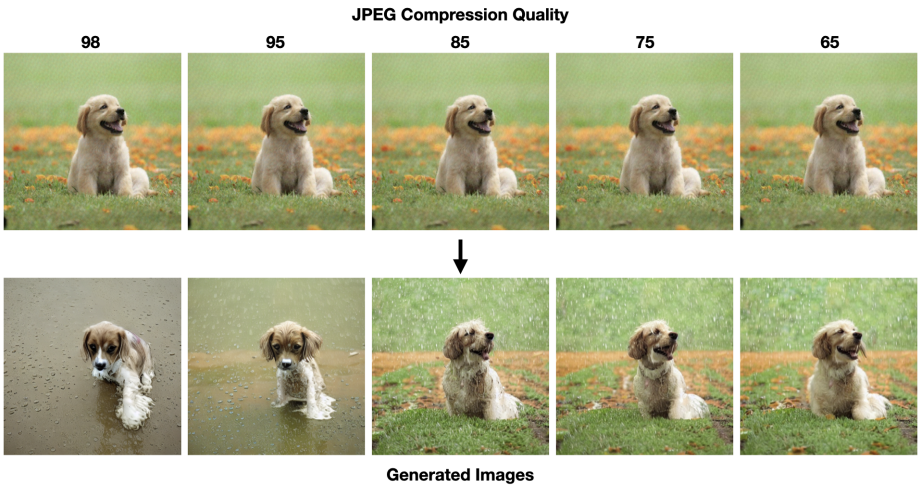


Figure 7.2: **More JPEG compression more reliably undermines photoguard protection.** First row: We take a photoguard (Encoder attack) image and JPEG compress it with varying quality. An image at 100% JPEG quality is almost equivalent to the original photoguard image, while 65% JPEG quality loses significant high-frequency information. Second row: Starting from the compressed image above, the adversary uses a diffusion model to make edits according to the same prompt setup as Figure 7.1. With more compression, the generated content better maintains the original subject. Between compression quality of 95% and 85%, enough of the photoguard noise is diminished, allowing stable image edits by an adversary.

they would choose not to edit the protected image. In this section, we demonstrate that if the adversary JPEG compresses the Photoguard-protected image, they can effectively perform edits using the same SDM used to protect the image. We also discuss the possibility of optimizing against this attack and analyze how other simple image transforms affect Photoguard protection.

7.2.1 JPEG Compression as a Low-Cost Transform

Given a source image (*e.g.* , an image of a dog in the grass) and a text prompt (*e.g.* , "dog under heavy rain and muddy ground real"), SDM can edit the source image to match the prompt. In Fig. 7.1, we show that when the Photoguard-protected image (from the Encoder attack) is used as the source, the edited image no longer maintains the original subject and the dog's front paw seems to blend with some kind of foam. However, if we JPEG compress the Photoguard image

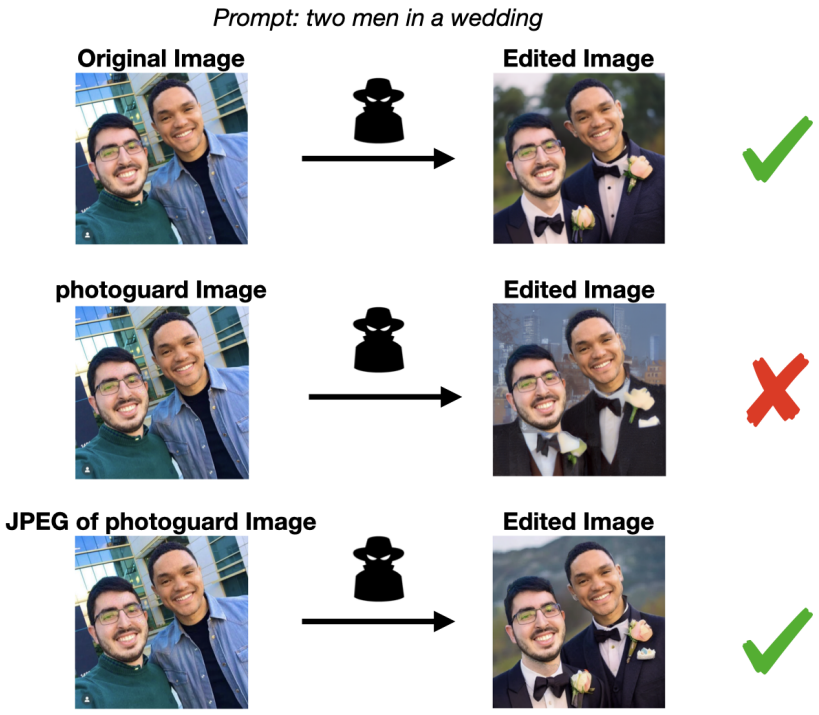


Figure 7.3: **JPEG compression allows an adversary to edit the background of a protected image found online.** First row: Given a text prompt, an adversary can make desired edits to an input image using a diffusion model. Second row: photoguard (Diffusion attack) [109] protects the original image before an adversary can access it by adding an imperceptible perturbation. When the adversary edits the photoguard image, the background is unrealistic. Third row: By JPEG compressing the photoguard image, an adversary can edit the photoguard image while maintaining the original subjects and adding key visual features of the text prompt.

with quality of 65, the edited image looks like the edit from the original, and is potentially even more faithful to the text prompt due to the more prominent rain.

To perform inpainting, SDM is conditioned on both a source image and a mask indicating the regions of the image that are to remain unedited. In the last row of Fig. 7.3, we show that an adversary can take a JPEG compressed Photoguard image (from the Diffusion attack) and generate a new image that contains more realistic background and clothing.

We explore the effect that JPEG compression of varying quality has on the resulting edited image in Fig. 7.2 for the Encoder attack and Fig. 7.4 for the Diffusion attack, finding that there

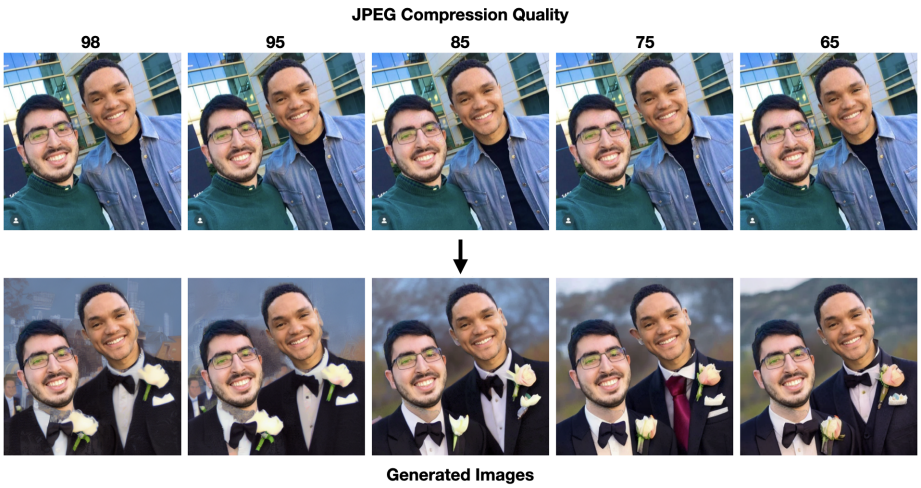


Figure 7.4: **More JPEG compression more reliably undermines photoguard protection.** First row: We take a photoguard (Diffusion attack) image and JPEG compress it with varying quality. Second row: Starting from the compressed image above, the adversary uses a diffusion model to make edits according to the same prompt setup as Figure 7.3. With more compression, the generated content background and clothing is more realistic. Between compression quality of 85% and 75%, enough of the photoguard noise is diminished, allowing stable image edits by an adversary.

exists a compression level range in which image editing performs well. If the Photoguard perturbations were mostly high-frequency, using a Gaussian blur should also be effective at restoring an adversary's editing abilities. However, in Sec. 7.2.2 we find that Gaussian and median blurs are ineffective. Other simple image transforms like rotations and flips (Sec. 7.2.3) are also ineffective. This is likely because the image encoder is mostly invariant to these transforms. While these simpler transforms were unsuccessful, we believe they should still be part of extensive robustness evaluations of future methods utilizing imperceptible perturbations. JPEG's ability to undermine the protective perturbation highlights the transforms' uniqueness in eliminating the perturbation while maintaining important semantic content.

Prior work has shown that JPEG could be used as a reasonable pre-processing step to defend against adversarial examples [104], but such pre-processing defenses are not as robust to

adaptive attackers, which go last in the classical game of adversarial examples. As pointed out in Salman et al. [109], this could imply that an approach like Photoguard could be improved to be robust against these transformations. Yet, we want to highlight that the game between attack and defense is *flipped* for data modifications, a modification like Photoguard has to anticipate future adaptive attacks that scrub it, and previous results on robustness of adversarial attacks might not apply.

7.2.2 Gaussian and Median Blurs of the Protected Image

Gaussian blurs are low-pass filters which attenuate high-frequency signals. If the most important component of a Photoguard perturbation were the high-frequency components, then a Gaussian blur should be successful at diminishing the protection. In Fig. 7.5, we show Gaussian blurs of Photoguard images do not enable SDM to generate reasonable images. Since Gaussian blurs also destroy a significant amount of semantic information, it is difficult to pinpoint whether the most important features of the Photoguard perturbations are high-frequency. We also experiment with median blurs in Fig. 7.6.

7.2.3 Rotations and Horizontal Flip of the Protected Image

The adversary need not edit the Photoguard-protected image as-is. Performing other simple transforms seems reasonable before editing the image. In Fig. 7.7 and Fig. 7.8, we experiment with rotating and flipping the protected image before using SDM. While the Photoguard perturbation was not optimized so that rotations and flips induce the same embedding from the encoder, generated images did not retain the original subject of the source image, suggesting the Photoguard

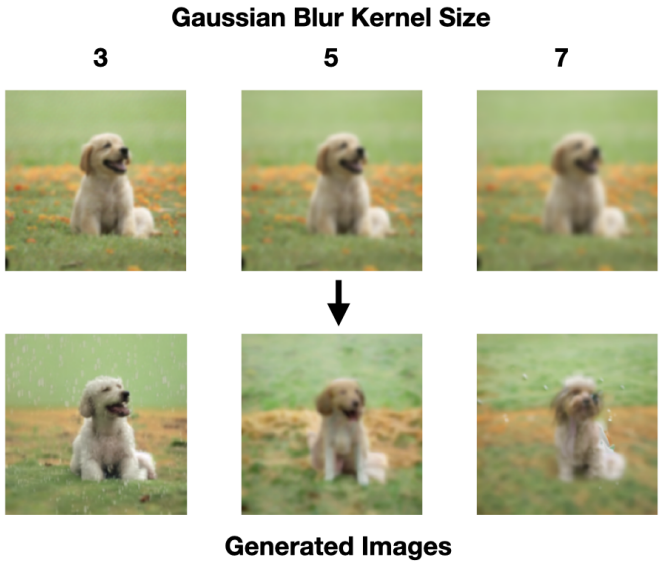


Figure 7.5: **Gaussian blurs of photoguard images**. First row: We perform a Gaussian blur on a photoguard image with varying kernel sizes. Second row: Starting from the blurred image above, the adversary uses a diffusion model to make edits according to the same prompt setup as Fig. 7.1. With a larger kernel, the generated images lose significant visual features and do not maintain the original subject, unlike JPEG compression.

perturbation was effective under these transforms.

7.3 Conclusion

Ultimately, the attack presented in this note is surprisingly simple, to the point where the (re-) encoding with different JPEG settings could also be considered just a benign editing operation, or a standard operation when storing the modified image. This makes it harder to verify how high the bar is really raised by adversarial noise image protection methods. Our results suggest that an adversary can bypass protections by JPEG compressing images that have been imperceptibly modified for protection. Protecting images from misuse remains an extremely difficult problem. As long as semantic information of an image is recognizable by a human, as is the case with imperceptibly perturbed images, the human visual system is a constructive proof

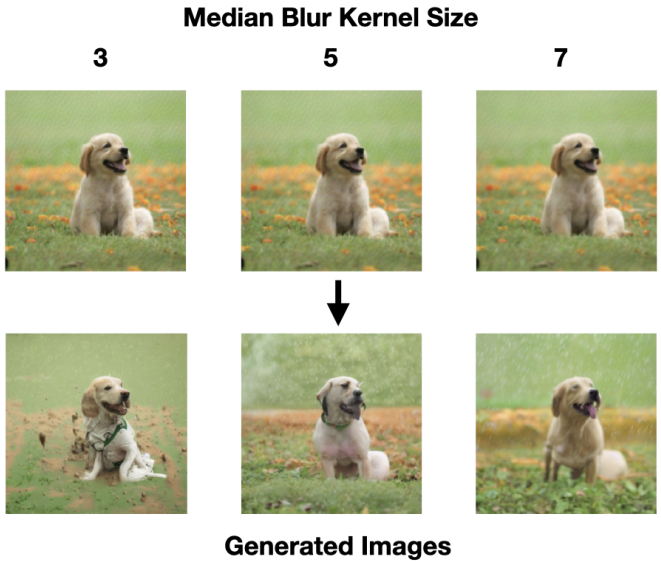


Figure 7.6: **Median blurs of photoguard images**. First row: We perform a median blur on a photoguard image with varying kernel sizes. Second row: Starting from the blurred image above, the adversary uses a diffusion model to make edits according to the same prompt setup as Fig. 7.1. With a larger kernel, the generated images seemingly begin to retain the original subject.

that there exists a function to recover the semantic information. Thus, for high-stakes settings where image misuse is of concern, imperceptible perturbations of any kind, regardless of the optimization details, are so far inadequate for protection against editing by an adversary.

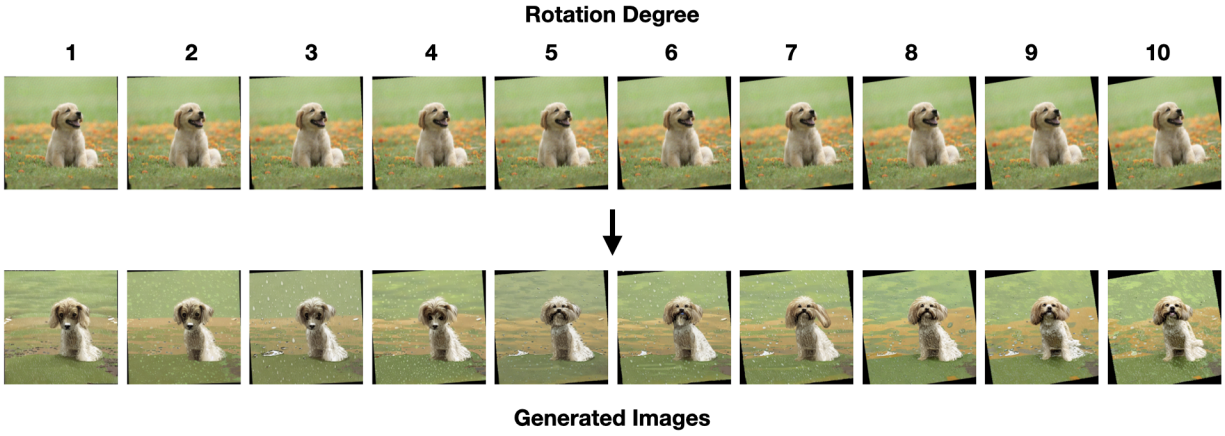


Figure 7.7: **Rotations of photoguard images**. First row: We perform counter-clockwise rotations of varying angles on a photoguard image. Second row: Starting from the rotated image above, the adversary uses a diffusion model to make edits according to the same prompt setup as Figure 7.1. The generated image does not retain the original subject in any of the rotations we tried.

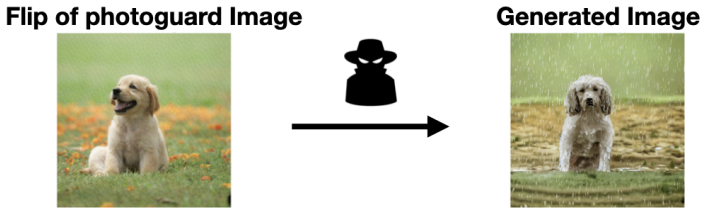


Figure 7.8: **Flip of photoguard image**. Left: We perform a horizontal flip on a photoguard image. Right: Starting from the flipped image, the adversary uses a diffusion model to make edits according to the same prompt setup as Figure 7.1. The generated image does not retain the original subject.

Part III

Unnecessary Computation During Inference

Chapter 8: Identifying and Evaluating Inactive Heads in Pretrained LLMs

Part I studied a learning preference that data can exploit. Part II studied the limits of that protection. This chapter studies a learned behavior that transformer models exploit but that also incurs unnecessary computation on every forward pass.

Attention is foundational to large language models (LLMs), enabling different heads to have diverse focus on relevant input tokens. However, learned behaviors like attention sinks, where the first token receives the most attention despite limited semantic importance, suggest some heads may be inactive, and point to a significant source of computational redundancy. To analyze this phenomenon, we evaluate 12 score functions that measure different ways a head can be inactive. Thresholding these scores allows us to analyze different sets of potentially inactive attention heads. We evaluate whether identified heads are inactive through model interventions, finding that more than 12% of attention heads are inactive on average, and can be ablated in specific contexts while maintaining MMLU accuracy to within 1% of the pretrained LLM. Across 3 model families, our score functions that measure the average norm of a head's output consistently identify inactive heads that would not have been found by score functions that rely solely on attention weights. We establish that relying on a score function that measures a first token attention sink would underestimate the prevalence of inactive heads, failing to identify more than 7% of inactive heads on average. We also show how measuring score distributions can provide

insights into attention behavior. For instance, we find evidence that finetuning causes little to no change in attention behavior, and that even within the same model family, large model scales present different attention behaviors. [1]

8.1 Introduction

Attention is a key component of the transformer architecture, which has led to breakthroughs in language modeling [61, 116, 117, 118, 119]. The attention mechanism allows tokens to incorporate information from relevant tokens, with multiple heads of attention capturing different types of relevance [59]. But several works have found that attention can become "dormant" and concentrate on initial tokens, which are semantically irrelevant [74, 120]. The question we seek to answer is: How prevalent are inactive attention heads?

Depending on how one defines the word *inactive*, different answers are possible. Past work has focused exclusively on the attention weights [72], labeling heads as "dormant" using a threshold-based score function. This is motivated by the idea that if a head attends primarily to the first token, and the first token has a near-zero value state, then the head output will be near-zero. However, this reasoning ignores other possibilities. Because an attention head's output is a convex combination of value vectors, a head could attend to multiple tokens with near-zero value states to produce a near-zero output. In this work, we explore multiple ways an attention head can be considered inactive by evaluating a range of score functions and determine the most robust method for identifying inactive heads.

While classifying heads using score functions offers insights into learned attention behaviors,

[1] This chapter follows "Identifying and Evaluating Inactive Heads in Pretrained LLMs" [115] by Pedro Sandoval-Segura, Xijun Wang, Ashwinee Panda, Micah Goldblum, Ronen Basri, Tom Goldstein, and David Jacobs. It was published at ICLR 2026.



Figure 8.1: **Score Functions for Inactive Heads.** Our simple score functions measure all three components of attention: attention weights, value vectors, and head output vectors. In each cell, we give an example of the type of attention head that would be identified by each score function once scores are thresholded. For attention weights, the colorbar displays weights. For vectors, color represents direction, and length of the blue bar represents magnitude. For example, the Avg Weight of First Token score [72, 121] is calculated by computing average weight to the first token, and heads that exhibit high attention to the first token are identified if their their score exceeds a threshold. Value vectors and head outputs do not play a role, which we illustrate with a fade. Including normalized versions of these 6 score functions, there are 12 in total.

simply identifying potentially inactive heads is not enough. We must also evaluate which set of identified heads best preserves model performance when such heads are zeroed out. Without zeroing out attention head outputs to erase their contribution to the hidden state, it is impossible to verify if identified heads are actually inactive. To address this, we perform model interventions while evaluating on the MMLU benchmark, and measure the effect of removing identified heads on model accuracy. Using 14 models across 3 model families, this experimental setup also allows us to analyze how attention head behaviors change as models scale and as models are finetuned. We make the following contributions:

- We evaluate 12 score functions that measure distinct properties of attention heads, and

we threshold scores to classify attention heads, based on different definitions of *inactive*. We evaluate whether identified heads are truly inactive through model interventions which show that, on average, more than 12% of attention heads are inactive in pretrained LLMs we consider. Using prior characterizations of "dormant heads" would underestimate inactive heads, and fail to identify 7% of heads that are inactive.

- We find that, across model families and across model scale, inactive heads are consistently identified through the same score functions. In particular, measuring the average head output norm is most indicative. By finding a more model-agnostic score function, we provide evidence that inactive heads across transformer LLMs should be understood through head outputs rather than attention weights.

- We demonstrate that analyzing score distributions provides useful insights into attention: they reveal that finetuning induces minimal changes in attention behavior, and that scaling has little effect until models reach large sizes. Together, this suggests attention head behavior is more invariant to common training modifications than one might expect.

While our work has the potential to be used for efficient inference, our focus is strictly on understanding inactive attention heads. Specifically, how can we identify them and evaluate whether they are truly inactive?

8.2 Background and Related Work

The idea that transformers may not be effectively utilizing all attention heads first came up in the context of machine translation. The work of Voita et al. [122] and Michel et al.

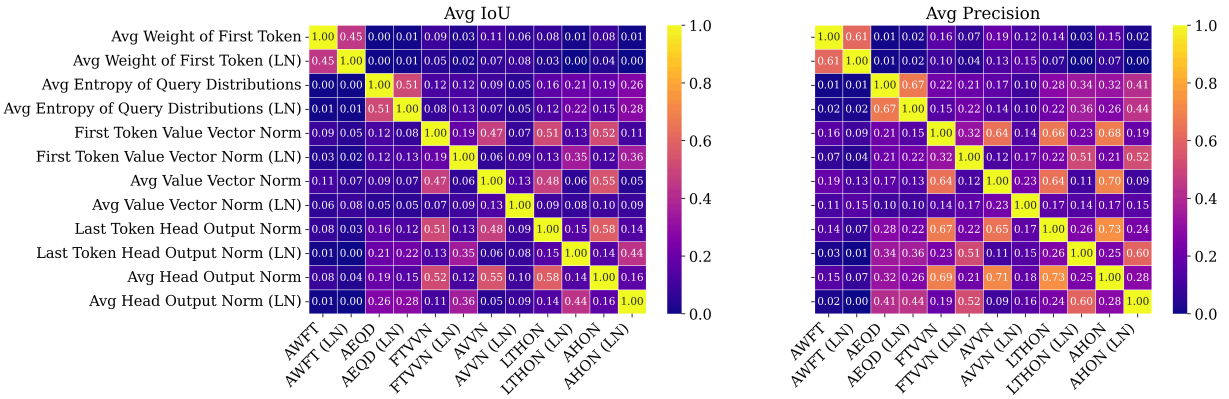


Figure 8.2: **Score functions identify different sets of heads**. Identifying different sets of heads ensures we capture a broad range of head characteristics, rather than focusing solely on the attention sink pattern of Avg Weight of First Token [121]. Using 100 FineWeb-Edu training samples, we measure IoU of classifications between each score function on Llama-3.1-8B. We also measure Precision of one score function's classifications relative to another's, using the column score function as ground truth. Column score functions are abbreviated using the first letter of each word, and are in the same order as rows. For each score function, we dynamically choose thresholds such that $\sim$10% of heads are identified as potentially inactive. Even head scores normalized by scores of other heads in the layer, denoted by "(LN)", do not show significant IoU or Precision with their unnormalized counterpart.

[123] demonstrated that attention heads in transformer models could be removed with minor performance degradations. Methods for determining which heads can be removed have involved optimization of different kinds of objectives based on: stochastic gates [122], importance scores [123], iterative pruning [124], and subset pruning [125]. In this chapter, however, we do not use static pruning, where a head is permanently removed. Rather, our model intervention study (Sec. 8.4.3) is a form of dynamic pruning, where different heads are dropped at every forward pass. Our objective is to measure how much head computation is wasted in each forward pass by inactive heads that can be zeroed out. We assume inactive heads are those that can be zeroed out, prior to concatenation and output projection of the multi-head attention mechanism.

Multi-head attention. The multi-head self-attention mechanism [59] allows multiple attention heads to operate in parallel, each focusing on different aspects of the input sequence. For an

input of N tokens with dimensionality d_m, learned linear projections transform queries, keys, and values ($\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{N \times d_m}$) into lower-dimensional representations: d_k-dimensional queries/keys and d_v-dimensional values, specific to each of the h heads. The self-attention operation within each head computes attention weights, defined as $\mathbf{A} = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}})$, where $\mathbf{A} \in [0, 1]^{N \times N}$ and each row sums to 1, which are then applied to the value vectors. Another way to view the output of the i^{th} attention head $\text{head}^i \in \mathbb{R}^{N \times d_v}$ is that, for every position in the sequence, we compute a convex combination of value vectors, weighted by the corresponding row of $\mathbf{A}$. The outputs of all heads are concatenated and passed through a final linear projection to produce the module's output. The attention weights, $\mathbf{A}$, are one of the few features we can visualize to get a sense of how pretrained LLMs work. For a token sequence, high attention weights (close to 1) reveal which tokens are considered most relevant. Thus, it is surprising that attention can concentrate on initial tokens, called "attention sinks", as shown in App. Fig. 8.19.

Attention sinks. An attention sink or sink token is a token that, despite limited semantic importance, disproportionally receives high attention weight from other tokens in a sequence. They tend to occur either at the first position of the input sequence [71, 72], at certain word tokens (*e.g.* , "and" and "of"), or delimiter tokens [73, 74]. The first token can be an attention sink even when it is not a BOS token [71, 121]. To understand the prevalence of attention heads with sinks, [121] propose to measure the average weight assigned to the first token and check if it exceeds a threshold τ: $\frac{1}{N}\sum_{i=0}^{N-1} \mathbf{A}_{i,0} > \tau$. In this chapter, we refer to this metric as "Avg Weight of First Token" and use it as a starting point, but ultimately find that different threshold-based score functions are needed for identifying inactive heads in model families outside of Llama [117] and GPT-2 [61].

As can be seen in App. Fig. 8.19, attention sinks also exhibit value-state drains [121, 126, 127], where the norm of the value state is near zero. Note that if the sink token's value state is zero but the attention weight to the sink token is one, the output of this head (prior to the output projection) is zero, leading Guo et al. [72] to call this a dormant attention head.

Dormant attention. Guo et al. [72] propose the idea that attention heads can be either active or dormant. They perform a model intervention study on 3 attention heads of Llama-2-7B Base, where a specific attention head output is zeroed out, and the difference in loss as a result of the intervention is measured. They find that the difference in loss can depend on whether input text is from Wikipedia or GitHub; a head that is dormant on Wikipedia samples does not change the loss when the head output is zeroed. In this chapter, we go beyond analyzing one attention head at a time and propose to zero out all "dormant heads" while evaluating models on a real-world benchmark task. If "dormant heads" are truly inactive, then zeroing them out should have little effect on model performance. We also propose new score functions that measure different definitions of *inactive*. Unlike Guo et al. [72] and Gu et al. [121] who measure only attention weights, our score functions also consider value vectors and head outputs. By considering the range of ways a head can be inactive, we present a more accurate picture of inactive attention heads in pretrained models. A better understanding of inactive attention heads has the potential to be used for efficiency [125], KV cache reduction [128], and compression [129].

8.3 Evaluating Score Functions for Inactive Heads

We outline a number of simple score functions, each capturing different ways a head could be considered inactive. Each score function assigns a score to every attention head in a

Table 8.1: **Equations of Score Functions.** We define how each score function is used to identify inactive heads with a threshold-based rule. Sample attention heads are in Sec. 8.4.7 Fig. 8.8. We sweep a range of thresholds τ for each function.

Score Function	**Definition**
Avg Weight of First Token (AWFT)	$\frac{1}{N}\sum_{i=0}^{N-1} \mathbf{A}_{i,0} > \tau$
Avg Entropy of Query Distributions (AEQD)	$\frac{1}{N}\sum_{i=0}^{N-1} \mathrm{Ent}(\mathbf{A}_{i,:}) < \tau$
First Token Value Vector Norm (FTVVN)	$\lVert\mathbf{V}_{0,:}\rVert_2 < \tau$
Avg Value Vector Norm (AVVN)	$\frac{1}{N}\sum_{i=0}^{N-1} \lVert\mathbf{V}_{i,:}\rVert_2 < \tau$
Last Token Head Output Norm (LTHON)	$\lVert\mathrm{head}^i_{N-1,:}\rVert_2 < \tau$
Avg Head Output Norm (AHON)	$\frac{1}{N}\sum_{i=0}^{N-1} \lVert\mathrm{head}^i_{i,:}\rVert_2 < \tau$

transformer LLM. By thresholding the scores, we classify and study different sets of heads. We summarize each score function in Tab. 8.1. We use the ℓ_2-norm wherever applicable.

Attention patterns. An attention head could appear inactive due to the patterns in its attention weights. Both overly focused and overly diffuse attention can be considered "inactive," depending on the input. Prior work has recognized the case where attention can concentrate on a single token [72, 121], where attention is distributed too little, and quantified it by calculating the Average Weight of the First Token (AWFT). However, this does not capture the case where there are multiple sink tokens (See App. Fig. 8.19). To measure when attention concentrates on a few tokens, we choose to measure the Entropy of each Query's Distribution over keys, averaged over all queries. Calculating this amounts to measuring entropy of each row of the attention weights, then taking an average. If average entropy of the head is low, it implies attention concentrates on few tokens. If a head's average entropy is under a threshold τ, we can classify it as potentially inactive. The smaller we make τ, the more strict we are in classifying heads that exhibit this pattern. Changing the direction of the inequality in Avg Entropy of Query Distributions could

capture cases where attention is too uniform (*i.e.* too much attention to all keys), but we chose not to consider this case because heads in early layers tend to exhibit this pattern, and early layers tend to be important [130].

Value vectors. An attention head could *appear* active with its attention patterns, but could be inactive if all the value vectors are near-zero. To measure this, we compute Avg Value Vector Norm: the average ℓ_2-norm of value vectors in the head. It is also possible that while attention patterns may not primarily concentrate on the first token, some tokens could still use the first token as a "value-state drain" [72]. To capture heads that do this, we compute the First Token Value Vector Norm, *i.e.* the ℓ_2-norm of the value vector corresponding to the first position. As with other score functions, we consider normalizing by the average score of other heads in the layer.

Head outputs. Finally, an attention head could be inactive if its output is small. This key assumption is most closely captured by the score functions that measure the head output: when a head produces small outputs, its contribution to the residual stream will be minimal, and these heads can be zeroed with little to no accuracy drop. Because multiple choice (MC) benchmark evaluations retrieve the probability assigned by the model to the correct answer from the probability distribution of the last position, we also propose to measure the Last Token Head Output Norm. To capture cases where most positions have head outputs that are small, we measure the Avg Head Output Norm. More information on why we do not use the circuit-based definition of a head output can be found in Sec. 8.4.6.

Normalization. We classify heads using the score functions in Tab. 8.1, but we also consider normalization by the score of other heads in the same layer. Abusing terminology, we refer

to this as layer normalization, or "(LN)" throughout. We do this because raw scores can vary dramatically across layers and models. For example, in Llama-3.2-3B, a value vector norm of 1.0 may be largest for a head in layer index 0, but smallest in layer index 27. Using a relative score allows us to make meaningful comparisons across layers and models. As an example, the Avg Head Output Norm (AHON) score function for the i^{th} attention head in a layer is computed as follows: $\text{AvgNorm}(\text{head}^i)$, where $\text{AvgNorm}(T) = \frac{1}{N}\sum_{i=0}^{N} \|T_{i,:}\|_2$ computes the average ℓ_2-norm of the rows of input matrix T and where $\text{head}^i \in \mathbb{R}^{N \times d_v}$ is the head output. Then, the normalized version, Avg Head Output Norm (LN), is simply normalized by the average AHON score of heads in the same layer:

$$\frac{\text{AvgNorm}(\text{head}^i)}{\frac{1}{N_{\text{layer}}}\sum_{j=0}^{N_{\text{layer}}} \text{AvgNorm}(\text{head}^j)} \tag{8.1}$$

Additional normalization strategies. For score functions that measure a particular token in the sequence, like Last Token Head Output Norm, it is also possible to normalize by other vectors within the same head. We refer to normalization by other vectors in the same head as head normalization, or "(HN)". We explored this normalization strategy with the Last Token Head Output Norm score function, but found it was always worse than the layer normalization, "(LN)". HN is only included in Figs. 8.10 to 8.12.

Thresholding the score of Eq. (8.1) whenever it is $< \tau$ controls how strictly we classify a head as inactive. For example, when $\tau = 0.1$, heads with output norms less than 10% of the layer average are considered inactive. Code implementations are provided in Sec. 8.6.9.

8.4 Experiments

First, we study which score function is best at identifying inactive heads. Then, we use head scores to study how attention behaviors change as models scale and are finetuned.

8.4.1 Setup

We download 14 pretrained models using Hugging Face `transformers` [131]: Llama-3.1-8B, Llama-3.1-8B-Instruct [117], Llama-3.2-3B, Llama-3.2-3B-Instruct [132], OLMo-2-1124-7B, OLMo-2-1124-7B-SFT, OLMo-2-1124-7B-DPO, OLMo-2-1124-7B-Instruct [118], Qwen2.5-0.5B, Qwen2.5-1.5B, Qwen2.5-3B, Qwen2.5-7B, Qwen2.5-7B-Instruct, and Qwen2.5-14B [119]. We refer to models from the same organization as belonging to the same "model family." We evaluate models on MMLU [133] (5-shot) in our model intervention experiments. For datasets that do not use an LLM-as-a-Judge, MMLU is the multiple-choice (MC) benchmark with the highest Spearman correlation with ChatBot Arena [134, 135], and thus makes it the most appropriate choice for aligning with real-world chatbot preferences while keeping evaluation consistent and tractable. Additional results on PIQA (0-shot) [136] and WinoGrande (5-shot) [137] can be found in Sec. 8.6.1. All evaluations are performed using `lm-evaluation-harness` [138]. We also use FineWeb-Edu [139], which is a pretraining dataset of educational webpages. We randomly truncate FineWeb-Edu training sequences to between 10 and 3000 tokens to exhibit a variety of sequence lengths. Additional model and dataset details are in Sec. 8.6.10.

Metrics and Thresholds. In a transformer with N_{heads} attention heads per N_{layers} layers, and a dataset $\mathcal{D}$ of token sequences, a forward pass on input sequence $x \in \mathcal{D}$ will produce attention

weights, value states, and head outputs for every head. We measure these components, and assign a score to every head, arranged in a matrix $\mathbf{S} \in \mathbb{R}^{N_{\text{heads}} \times N_{\text{layers}}}$. Thresholding $\mathbf{S}$ results in a boolean matrix $\mathbf{B}$ of the same size. In our model intervention experiments, a different boolean matrix is constructed for every new forward pass and "% of Model Heads Zeroed" refers to the percent of `True` values in $\mathbf{B}$, averaged over all token sequences in $\mathcal{D}$. The proportion of heads each score function identifies as inactive can be controlled by the threshold τ, which we vary for each function. The thresholds for each score function are chosen by measuring head scores across MMLU inputs, constructing the CDF of scores, and calculating the p-th quantile[2] for $p \in [0, 5, 10, 15, 20, 25, 30]$. This way, we can estimate the proportion of heads that will be selected for each threshold, allowing us to focus on zeroing out at most 30% of heads. All score distributions are in App. Fig. 8.16.

8.4.2 Score Functions Identify Different Sets of Heads

To ensure that each score function measures a different aspect of attention heads, we measure the agreement between predictions (post-thresholding) using IoU. We also measure precision, taking one of the score functions as ground truth. We use Llama-3.1-8B and 100 FineWeb-Edu sequences to collect scores, and we threshold them dynamically to make predictions. In Fig. 8.2, we see that the max IoU is 0.58, while the max Precision is 0.73. This suggests no score function is effectively the same as another. High agreement tends to occur between unnormalized and normalized pairs of score functions (*i.e.* AWFT and AWFT (LN)), but not always. For example, Last Token Head Output Norm has an IoU of 0.58 with Average Head

[2] In the case of Avg Weight of the First Token, we calculate the $(1-p)$-th quantile because the threshold inequality reversed.

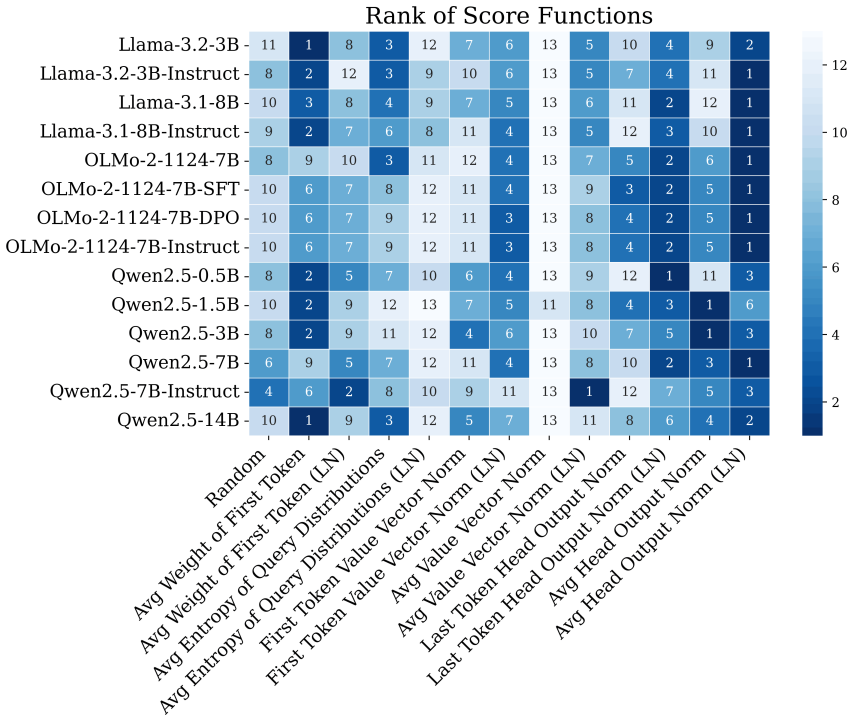


Figure 8.3: **Measuring Avg Head Output Norm is best at identifying inactive heads for most models**. For every model, we rank the 12 score functions by normalized AUC, which captures their ability to select inactive heads. Avg Head Output Norm (LN) ranks 1st for 8 out of 14 models, and ranks in top-3 for 13 out of 14 models. Top score functions are consistent across model families.

Output Norm. In a similar vein, First Token Value Vector Norm has an IoU of 0.47 with Average Value Vector Norm. This effect may occur because the presence of small-magnitude vectors drives down the overall average magnitude. The score function that best predicts the same heads as AWFT is AWFT (LN) with a precision of 0.61. Notably, normalizing by the average score of other heads in the layer significantly reorders heads. All other score functions have Precision under 0.19 with AWFT. Because each score function identifies a distinct subset of heads, we can verify their predictions knowing that each captures different underlying properties of attention heads.

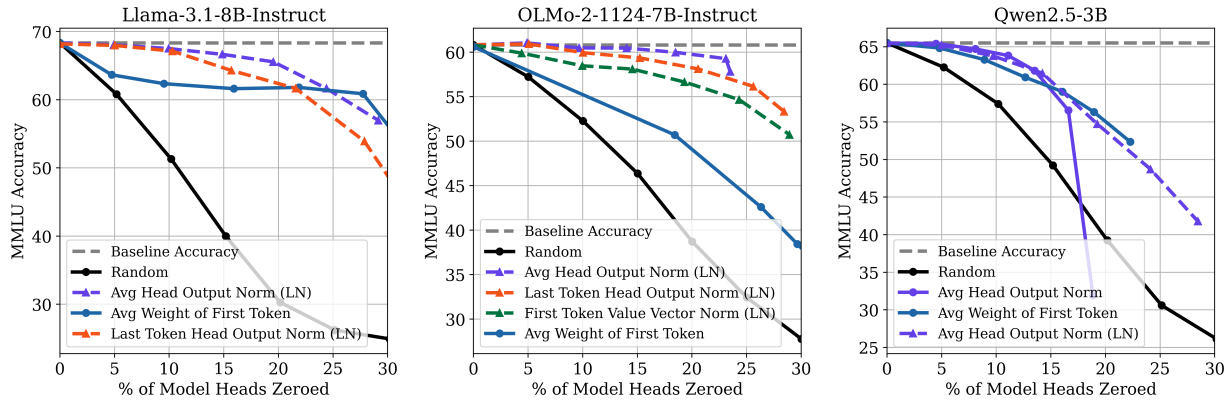


Figure 8.4: **Inactive heads can be identified and zeroed with minor performance degradation.** For each model, we plot the top-3 score functions. Average Weight of First Token is not in the top-3 for OLMo-2-1124-7B-Instruct, but we include it for comparison to prior work. The gray dotted line represents baseline accuracy of the pretrained model. The black line represents zeroing out heads uniformly at random. The Avg Head Output Norm (LN) score function is best at identifying the most heads, that when zeroed, maintain accuracy. Complete results for all models and score functions can be found in App. Fig. 8.10.

8.4.3 Verifying Inactive Heads through Model Interventions

To verify inactive attention heads, we conduct model interventions by zeroing out the outputs of identified heads from each score function, and measuring the resulting impact on model accuracy. For every model, we also compare to a random baseline where attention heads are zeroed uniformly at random. An effective inactive head score function should identify a substantial fraction of heads that can be zeroed out without compromising the performance of the pretrained model. What score function is best at identifying inactive attention heads?

In Fig. 8.4, we measure how MMLU accuracy changes as we zero out heads selected by different score functions. For brevity, we present one model from each model family: Llama-3.1-8B-Instruct, OLMo-2-1124-7B-Instruct, and Qwen2.5-3B. To do this, we vary the threshold of each score function such that identified heads are at most 30% of the model's heads. We plot the top-3 score functions for every model in Fig. 8.4, but complete results for all models and score

Table 8.2: **On average, more than 12% of attention heads can be zeroed while keeping average accuracy within 1% of the original model.** For every model, we calculate the highest percentage of heads that can be zeroed using Average Weight of First Token (AWFT), and compare to our score functions (*i.e.* all score functions excluding AWFT). Higher is better. The overall best score function is also shown. Our score functions only improve or are within less than 0.3% of AWFT. We are able to identify and verify more inactive heads than previously possible with AWFT. Specifically, using AWFT would classify less than 5% of heads as inactive, which misses 7% of heads, on average.

Model	% of Heads Zeroed		Best Score Function
	AWFT	Ours	
Llama-3.2-3B	8.34	8.05 (-0.29)	Avg Weight of First Token
Llama-3.2-3B-Instruct	6.87	13.04 (+6.17)	Avg Head Output Norm (LN)
Llama-3.1-8B	8.56	17.11 (+8.55)	Avg Head Output Norm (LN)
Llama-3.1-8B-Instruct	1.01	10.97 (+9.95)	Avg Head Output Norm (LN)
OLMo-2-1124-7B	0.42	8.34 (+7.93)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-SFT	1.70	18.95 (+17.25)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-DPO	2.14	20.60 (+18.46)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-Instruct	1.46	19.54 (+18.07)	Avg Head Output Norm (LN)
Qwen2.5-0.5B	7.43	14.42 (+6.99)	Last Token Head Output Norm (LN)
Qwen2.5-1.5B	7.55	7.49 (-0.07)	Avg Weight of First Token
Qwen2.5-3B	5.67	8.78 (+3.11)	Avg Head Output Norm
Qwen2.5-7B	1.25	7.54 (+6.29)	Avg Head Output Norm (LN)
Qwen2.5-7B-Instruct	1.13	5.76 (+4.63)	Avg Value Vector Norm (LN)
Qwen2.5-14B	11.04	9.88 (-1.17)	Avg Weight of First Token
Average	4.61	12.18 (+7.56)	-

functions can be found in App. Fig. 8.10. In Tab. 8.2, we quantify the percent of heads that can be zeroed while keeping accuracy to within 1% of baseline. We find that, for 8 out of 14 models, the Avg Head Output Norm (LN) score function is best at identifying the most attention heads that when zeroed, maintain accuracy to within 1% of baseline. For 13 out of 14 models, AHON (LN) ranks in the top-3 score functions. Avg Weight of First Token underestimates the number of inactive heads.

The fact that a single, simple threshold-based function like Avg Head Output (LN) can

identify the most inactive heads for multiple models, suggests that looking at attention weights is a misleading signal. While heads with attention sink patterns seem to occur in all models we consider (See App Fig. 8.19), our results suggest these patterns are not solely indicative of inactivity. Moreover, our results confirm that while attention head patterns may appear "dormant" [72], their outputs are not. Removing attention heads using Avg Weight of First Token as the score is particularly ineffective for OLMo-2 models. While it is appealing to look only at attention patterns as a consistent feature across models of varying organizations and architectures, it appears that small head outputs are consistently more indicative of inactivity.

Despite looking for negligible head outputs, the complexity and non-linearity of LLMs make it difficult to know whether heads with "small" outputs can be zeroed while maintaining accuracy. To the best of our knowledge, we are the first to systematically evaluate the degree of head inactivity and investigate whether these heads are unnecessary. **Takeaway:** Inactive attention heads are best identified by measuring Avg Head Output Norm (LN). Measuring attention weight patterns like Avg Weight of First Token are not model-agnostic.

Ranking Score Functions. Each of 12 score functions use a different set of 7 thresholds. Accuracy curves that terminate at different x-axis values make it difficult to assess which method is best. So, we choose to measure performance by Area Under the Curve (AUC) normalized by the x-axis span of each accuracy curve. For every model, the ranking of each score function is displayed in Fig. 8.3. Notably, Avg Head Output Norm and its normalized version are more model-agnostic, ranking 1^{st} for 10 out of 14 models. In contrast, Avg Weight of First Token ranks 1^{st} for only Llama-3.2-3B and Qwen2.5-14B, and performs most poorly for the OLMo-2 family of models.

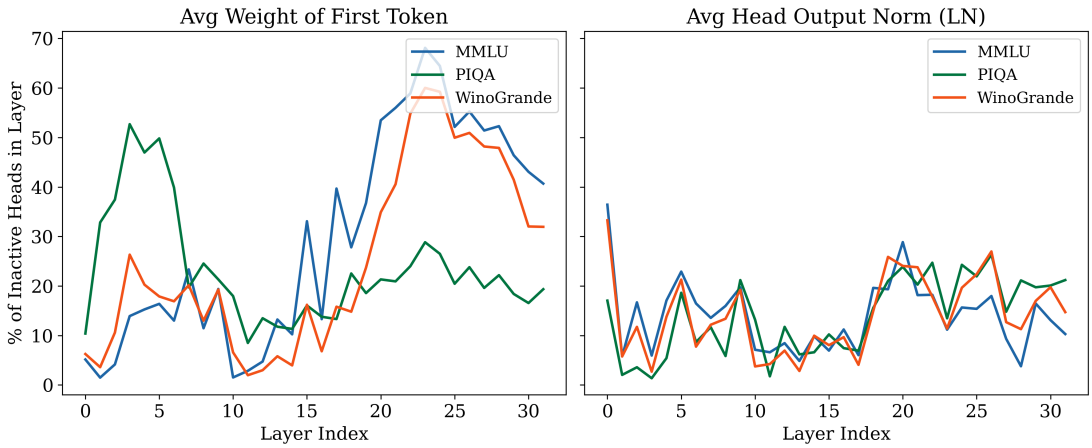


Figure 8.5: **Average Head Output Norm is a stable measure layer-wise inactivity.** Layer-wise percentage of inactive heads for OLMo-2-1124-7B-Instruct across MMLU, PIQA, and WinoGrande. (Left) Identifying inactive heads via Average Weight to First Token (AWFT) shows high sensitivity to the dataset. (Right) In contrast, using Average Head Output Norm (LN) yields more stable, dataset-agnostic percentages of layer inactivity.

8.4.4 Inactive Heads by Layer

To better understand the poor performance of Average Weight of First Token (AWFT) on the OLMo-2 family, we investigate the stability of two score functions on different data distributions. Using OLMo-2-1124-7B-Instruct, we evaluate 3 datasets: MMLU [133], PIQA [136], and WinoGrande [137], and plot the layer-wise percentage of identified inactive heads in Fig. 8.5. We compare Average Weight of First Token (AWFT) [121] against Average Head Output Norm (LN), the best performing method in Sec. 8.4.3.

To ensure a fair comparison, we calibrate thresholds for both score functions such that 15% of heads are identified as potentially inactive. Notably, achieving this target classification with AWFT requires highly disparate thresholds depending on the task ($\tau_{\text{MMLU}} = 0.077$, $\tau_{\text{PIQA}} = 0.265$, $\tau_{\text{WinoGrande}} = 0.109$). Conversely, the thresholds required for AHON (LN) are more consistent across datasets ($\tau_{\text{MMLU}} = 0.457$, $\tau_{\text{PIQA}} = 0.435$, $\tau_{\text{WinoGrande}} = 0.473$), indicating a

higher degree of generalizability.

The instability of AWFT is further evidenced by the layer-wise percentage of inactive heads. When using AWFT (Fig. 8.5, Left), the per-layer inactive percentages exhibit high sensitivity to the dataset. For instance, on MMLU, inactive heads are concentrated in later layers (peaking above 60% in layer 23). On PIQA, however, large fractions of inactive heads are in early layers (above 50% in layer 3). In contrast, identifying inactive heads via AHON (Fig. 8.5, Right) mitigates this variance and shows stable inactive percentages across all three datasets. Measuring Avg Head Output Norm (LN) provides a dataset-agnostic measure of head inactivity.

8.4.5 Score Distributions for Studying Attention Behaviors

As described in Sec. 8.3, each score quantifies a characteristic about an attention head: how much attention is paid to the first token, how small is the first value vector, how small is the average head output, etc. Analyzing the scores of heads can give us a broad idea of attention head behavior. Every model and score function combination produces a distribution of scores. By measuring the similarity of these score distributions, we can determine whether attention behavior is similar or different. With the exception of the AWFT score function, most score distributions have similar modes (See App. Fig. 8.16). Thus, we chose the Wasserstein distance to measure similarity between head score distributions. In Fig. 8.6, we see that AWFT score distributions are similar within a model family, illustrated by a dark block diagonal pattern. The only outlier is Qwen2.5-14B, which has distributions more similar to Llama models than Qwen2.5 models. For AHON (LN) score distributions, Llama and OLMo-2 models are more similar to each other than to Qwen2.5 models.

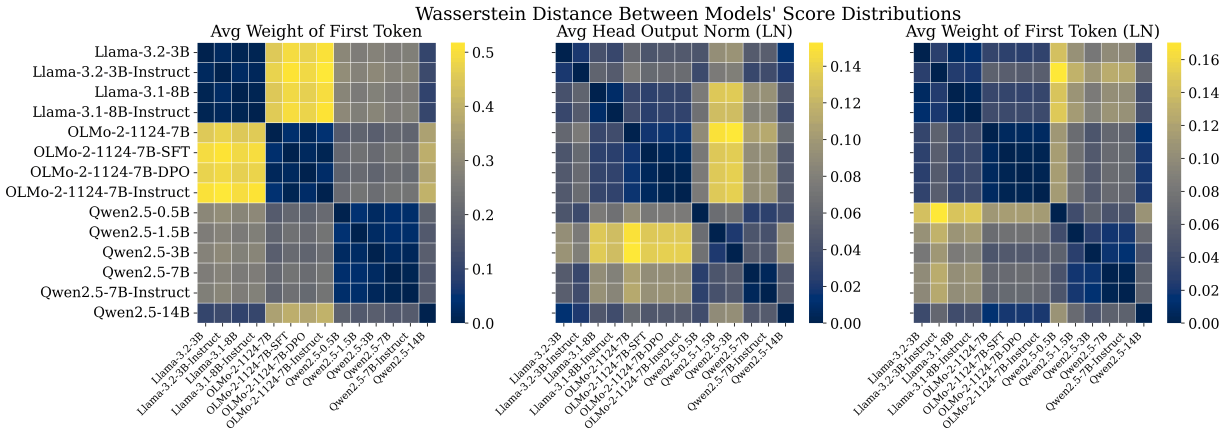


Figure 8.6: **Head score distributions can be similar within a model family**. Smaller Wasserstein distances indicate similar score distributions. Full results for all 12 score functions can be found in App. Fig. 8.17.

We use the Qwen2.5 models [119] to understand how model scale affects attention head behaviors. There are 5 model types ranging from 0.5B parameters to 14B parameters, but while the models share the same pre-training data, the exact number of training tokens each model was exposed to is not publicly documented. Assuming model scale is the only variable, it appears that model scale impacts attention head behaviors. As Qwen2.5 models grow in size, they are quite similar to one another, until the 14B scale. For example, on most score functions, Wasserstein distance is near-zero, indicated by a clear dark region that corresponds to inter-model similarity. But at the 14B scale, there appears to be only 2 score function distribution where Qwen2.5-14B is similar to other models in the Qwen2.5 family: Avg Value Vector Norm and its layer normalized version. This may suggest that larger models learn to specialize their heads in different ways. The similarity of score distributions between the Llama-3 models we consider may have more to do with their training objective as opposed to training data. In particular, because Llama-3.2 was created from pruning Llama-3.1-8B, then performing knowledge distillation, these models are *trained* to be similar [132].

To study finetuning, we primarily rely on the OLMo-2 models which have checkpoints after SFT, after DPO [140], and after RLHF [141]. We also have the Instruct models corresponding to the Llama-3.2-3B, Llama-3.1-8B, and Qwen2.5-7B base models. Figs. 8.6 and 8.17 illustrate that all finetuning strategies we consider (SFT, DPO, RLHF) have nearly no effect on score distributions, suggesting little change in attention behaviors. Across all score functions, finetuned models have the smallest Wasserstein distances with their base model. In the case of all the OLMo-2 models, the Wasserstein distances of their score distributions are more similar to each other than any other model. These results indicate that finetuning primarily preserves the underlying attention head behaviors of the base model.

8.4.6 Circuit Definition of Attention Head Output

Vaswani [59] define an attention head output as the convex combination of value states, resulting in a d_v-dimensional vector where $d_v = \frac{d_{\text{model}}}{h}$, d_{model} is the hidden size, and h is the number of heads. Other works multiply the d_v-dimensional vector by a $\mathbb{R}^{d_v \times d_{\text{model}}}$ slice of the output projection matrix [69]. This is often called the "circuit-based" definition. In the following, we explain why this choice does not affect our intervention results.

Given N input tokens, an attention head will multiply attention weights $A_i \in \mathbb{R}^{N \times N}$ with value states $V_i \in \mathbb{R}^{N \times d_v}$ so that $Z_i = A_i V_i \in \mathbb{R}^{N \times d_k}$ for $i = 1 \dots h$, where i is the head index. In multi-head attention, we first concatenate $H = [Z_1; \dots; Z_h] \in \mathbb{R}^{N \times (hd_v)}$. Then, do an output projection

$$Y = HW_O \quad \text{with} \quad W_O \in \mathbb{R}^{(hd_v) \times d_{\text{model}}}$$

Elhage et al. [69] partition W_O row-wise into h blocks of size $d_v \times d_{\text{model}}$:

$$W_O = \begin{bmatrix} W_O^{(1)} \\ W_O^{(2)} \\ \vdots \\ W_O^{(h)} \end{bmatrix}, \quad W_O^{(i)} \in \mathbb{R}^{d_v \times d_{\text{model}}} \tag{8.2}$$

Then they note that the output can be written exactly as the sum of h terms:

$$Y = \sum_{i=1}^{h} Z_i W_O^{(i)} \tag{8.3}$$

Elhage et al. [69] use the above to argue that attention heads are independent saying that this is "equivalent to running heads independently, multiplying each by its own output matrix, and adding them into the residual stream." But crucially, before we add the output of the attention module to the residual stream, we actually add up all head contributions (*i.e.* the summation in Eq. (8.3)). This means that W_O can implement cancellations (if, for example, two head contributions sum to zero) or other special linear combinations of heads.

Our choice and why it does not affect results in Sec. 8.4.3. Because W_O mixes all heads' outputs into the shared model space, we did not want to integrate these parameters into our measurement. Thus, we define the pre-output-projection Z_i as the attention head output. Importantly, whether we choose to define Z_i or $Z_i W_O^{(i)}$ as the i^{th} head's output, when "zero out a head" Z_i, both will be zero.

Implementing the circuit-based definition. We implement the circuit-based definition of Average Head Output Norm (LN), and evaluate it on MMLU using 3 models in Fig. 8.7. More specifically, this means that for every head of index i we measure the norms of the rows of $Z_i W_O^{(i)} \in \mathbb{R}^{N \times d_{\text{model}}}$,

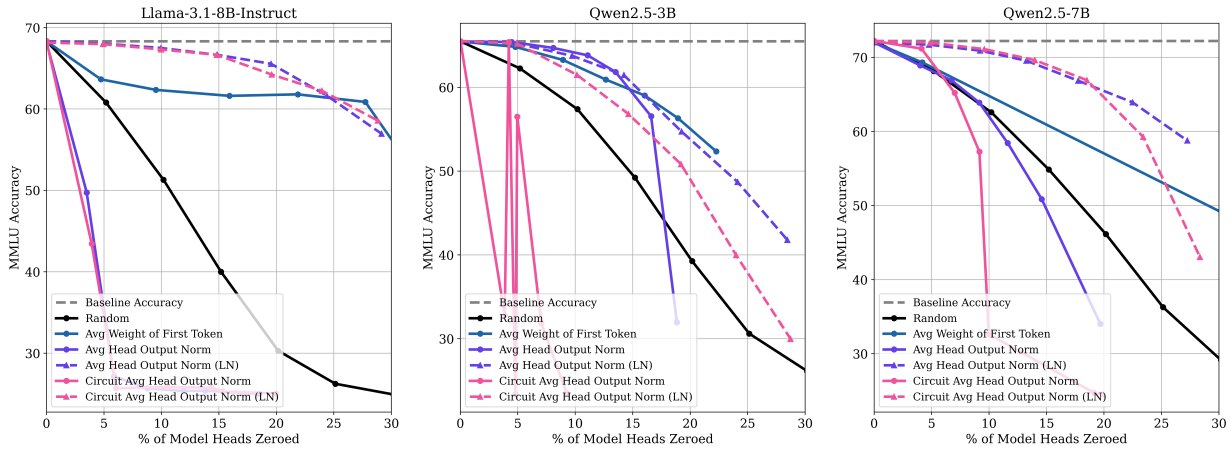


Figure 8.7: **Circuit-based definition of head output**. For 3 different models evaluated on MMLU, using the normalized circuit-based definition of a head output is comparable to using the original definition. Dashed lines denote layer normalized (LN) score functions.

Table 8.3: **Circuit-based definition of head output.** We measure the percent of model heads that can be removed while maintaining accuracy within 1% of baseline (higher is better). For 3 different models evaluated on MMLU, using the circuit-based definition of a head output is comparable to using the original definition.

Model	AWFT	AHON (LN)	Circuit-based AHON (LN)
Llama-3.1-8B-Instruct	1.01	**10.97**	9.88
Qwen2.5-3B	5.67	**7.29**	5.95
Qwen2.5-7B	1.25	7.54	**9.04**
Average	2.64	**8.60**	8.29

then average them to produce a score. Tab. 8.3 quantitatively shows that the percent of heads that can be zeroed using both definitions is comparable.

8.4.7 Sample Inactive Heads Identified by Score Functions

In Fig. 8.8, we give examples of attention heads that are identified by each unnormalized score function, by using threshold-based rules of Tab. 8.1. To keep each attention weights matrix small, we use a random, short 50 token sequence from FineWeb-Edu and execute a forward pass

on Qwen2.5-7B and OLMo-2-1124-7B. In Fig. 8.9, we show how the normalized AHON (LN) score function can capture distinct kinds of heads. For example, the first head has a first token sink, the second head has two sinks, and the third head has a second token sink. AHON (LN) is able to identify all three as inactive.

As a summary of the kind of head that each score function is meant to capture, we provide a short explanation for each:

- Avg Weight of First Token (AWFT): When attention concentrates on first token
- Avg Entropy of Query Distributions (AEQD): When attention concentrates on a few tokens
- First Token Value Vector Norm (FTVVN): When first value vector is near-zero
- Avg Value Vector Norm (AVVN): When most value vectors are near-zero
- Last Token Head Output Norm (LTHON): When last head output token near-zero
- Avg Head Output Norm (AHON): When most head output tokens are near-zero

8.5 Conclusion

Understanding when attention heads are inactive requires looking beyond attention weights. Simple score functions, based on previously ignored components like value vectors and head outputs, can capture a wider array of inactivity. By assigning different scores to every head in a transformer LLM, we can classify different subsets of attention heads as inactive using a threshold. While there is some overlap in the subsets of inactive heads we classify using different score functions, we find that most score functions capture diverse characteristics. By zeroing out

identified inactive heads during MMLU evaluations, we find that more than 12% of heads can be zeroed out while maintaining accuracy to within 1% of the baseline model, on average. Using a score function based on a first token attention sink would classify less than 5% of heads as inactive on average, which underestimates the true prevalence of inactive heads. For the majority of models we consider, the score function that measures average head output norm produces the best signal for identifying inactive heads across model families. While attention sinks and sink tokens are almost synonymous with an inactive head, our work demonstrates that there are other sets of heads that are more inactive, and that we can identify them in a more model-agnostic way. Additionally, we show how measuring score distributions can be informative: we use them to explore how finetuning causes little to no change to attention behavior, and how model scale does very little to attention behavior until very large scale. Future work could consider how the MLP module, which follows the attention module, could be inactive per-token if converged to an optimal hidden state.

8.6 Appendix

8.6.1 Full Results of Sec. 8.4.3

In Sec. 8.4.3 Fig. 8.4, we present intervention results for 3 models. Results for all 14 models are presented in Fig. 8.10.

8.6.1.1 Additional Datasets

We also run the same experiments and analysis on the PIQA (0-shot) [136] and WinoGrande (5-shot) [137] datasets. Plots on how accuracy degrades with different percent of model heads

zeroed can be found in Figs. 8.11 and 8.12. The quantitative analysis, where we measure the maximum proportion of heads that we can zero while maintaining accuracy to within 1% of the original model can be found in Tabs. 8.4 and 8.5. On PIQA, our score functions show that more than 21% of attention heads can be zeroed while keeping average accuracy within 1% of the original model. On WinoGrande, more than 14% of attention heads can be zeroed while keeping average accuracy within 1% of the original model. Thus, our claim in this chapter is a conservative estimate of inactive heads. In all cases, our score functions can identify more heads than Avg Weight of First Token (AWFT).

Table 8.4: **PIQA Benchmark: On average, more than 21% of attention heads can be zeroed while keeping average accuracy within 1% of the original model.** For every model, we calculate the highest percentage of heads that can be zeroed using Average Weight of First Token (AWFT), and compare to our score functions (*i.e.* all score functions excluding AWFT). Higher is better. The overall best score function is also shown. We are able to identify and verify more inactive heads than previously possible with AWFT.

Model	% of Heads Zeroed		Best Score Function
	AWFT	Ours	
Llama-3.2-3B	21.76	17.37 (-4.39)	Avg Weight of First Token
Llama-3.2-3B-Instruct	15.32	25.78 (+10.46)	Avg Head Output Norm (LN)
Llama-3.1-8B	0.00	17.85 (+17.85)	First Token Value Vector Norm (LN)
Llama-3.1-8B-Instruct	31.30	25.39 (-5.91)	Avg Weight of First Token
OLMo-2-1124-7B	20.74	22.70 (+1.97)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-SFT	1.87	20.12 (+18.26)	Last Token Head Output Norm (LN)
OLMo-2-1124-7B-DPO	2.93	15.87 (+12.94)	First Token Value Vector Norm (LN)
OLMo-2-1124-7B-Instruct	2.54	26.83 (+24.30)	Avg Head Output Norm (LN)
Qwen2.5-0.5B	0.00	18.36 (+18.36)	Avg Head Output Norm (LN)
Qwen2.5-1.5B	24.33	21.32 (-3.01)	Avg Weight of First Token
Qwen2.5-3B	16.51	21.42 (+4.91)	Avg Head Output Norm
Qwen2.5-7B	0.00	28.66 (+28.66)	First Token Value Vector Norm
Qwen2.5-7B-Instruct	2.84	11.45 (+8.61)	Avg Head Output Norm (LN)
Qwen2.5-14B	0.00	26.12 (+26.12)	First Token Value Vector Norm (LN)
Average	10.01	21.38 (+11.37)	-

Table 8.5: **WinoGrande Benchmark: On average, more than 14% of attention heads can be zeroed while keeping average accuracy within 1% of the original model.** For every model, we calculate the highest percentage of heads that can be zeroed using Average Weight of First Token (AWFT), and compare to our score functions (*i.e.* all score functions excluding AWFT). Higher is better. The overall best score function is also shown. We are able to identify and verify more inactive heads than previously possible with AWFT.

Model	% of Heads Zeroed		Best Score Function
	AWFT	Ours	
Llama-3.2-3B	9.11	11.64 (+2.53)	Avg Head Output Norm (LN)
Llama-3.2-3B-Instruct	5.57	20.37 (+14.79)	Avg Head Output Norm (LN)
Llama-3.1-8B	6.86	17.08 (+10.22)	Avg Head Output Norm (LN)
Llama-3.1-8B-Instruct	12.44	11.89 (-0.55)	Avg Weight of First Token
OLMo-2-1124-7B	15.39	16.12 (+0.73)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-SFT	2.32	19.00 (+16.68)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-DPO	2.79	16.94 (+14.15)	Avg Head Output Norm (LN)
OLMo-2-1124-7B-Instruct	3.02	20.91 (+17.89)	Last Token Head Output Norm (LN)
Qwen2.5-0.5B	6.49	11.35 (+4.86)	Avg Head Output Norm (LN)
Qwen2.5-1.5B	7.49	12.68 (+5.18)	First Token Value Vector Norm
Qwen2.5-3B	5.73	10.32 (+4.60)	First Token Value Vector Norm
Qwen2.5-7B	6.60	15.69 (+9.10)	First Token Value Vector Norm (LN)
Qwen2.5-7B-Instruct	2.00	8.03 (+6.03)	Avg Value Vector Norm (LN)
Qwen2.5-14B	10.80	10.25 (-0.55)	Avg Weight of First Token
Average	6.90	14.45 (+7.55)	-

All models and score function evaluations can be found in Fig. 8.10.

8.6.2 Multiple Forward Pass Analysis

The main body of the chapter focuses on analyzing a single forward pass because that is how log-likelihood evaluations of MMLU, PIQA, WinoGrande and other MC benchmarks are performed. To explore how heads may change from inactive to active, based on increasing context, we choose 3 FineWeb-Edu train sequences of at least 512 OLMo-2 tokens in length:

- Seq 0 ID: urn:uuid:0d8a309d-25c5-405d-a08a-c11239f0d717. Category: Literary commentary.

- Seq 1 ID: urn:uuid:316c7af5-14e1-4d0b-9576-753e17ef2cc5. Category: Popular Science or Journalism.
- Seq 2 ID: urn:uuid:c337bcd8-6aa1-4f2d-8c48-b916442ebbee. Category: Educational article on software licensing.

We choose to evaluate inactive heads of OLMo-2-1124-7B-Instruct as each of these sequences grows in Fig. 8.13. We analyze AWFT and AHON (LN) score functions because AWFT is prior work, and AHON (LN) is the best score function we discover in Sec. 8.4.3. As expected, AWFT presents decaying curves where the model has fewer inactive heads as sequence length increases. As sequences increase in length, each query has more potential tokens it can attend to, making it less likely that the first token receives disproportionate attention. Additionally, as can be seen in Fig. 8.20 and Sec. 8.4.7, OLMo-2 models tend to use additional sink tokens (not just the first token). In contrast, measuring AHON (LN) inactive heads as sequence length increases, we observe noisy measurements of inactive head proportions in short sequences that converge as each sequence gets sufficiently long. In measuring both AWFT and AHON (LN) inactive heads, it is clear that inactivity is context-dependent.

We are also interested in where changing attention heads are located. We examine inactive head masks for AWFT and AHON (LN) in Figs. 8.14 and 8.15. Every matrix cell indicates a particular attention head, and we color yellow any head that is identified as inactive. Unlike AWFT score function measurements, AHON (LN) inactive heads are more spread out throughout the model. We also observe the same phenomenon in Sec. 8.4.4 Fig. 8.5.

8.6.3 Score Distributions on MMLU

All score function distributions, from MMLU, can be found in Fig. 8.16.

8.6.4 Full Results of Sec. 8.4.5 Wasserstein Distances

All Wasserstein distance matrices, for all score functions, can be found in Fig. 8.17.

8.6.5 Score Distributions Comparison: FineWeb-Edu vs. MMLU

In Fig. 8.18, we show how the distribution of Avg Weight of First Token scores change when the input dataset changes. Qwen2.5-7B has a small fraction of heads that exceed 0.8 average attention to the first token on FineWeb-Edu, but on MMLU there are essentially none. One explanation is the token sequence length which, for 5-shot MMLU, tend to be approximately 3000 tokens long. In contrast, the FineWeb-Edu sequences we use are randomly truncated to be between 30 and 3000 tokens.

8.6.6 Attention sinks and value-state drains

Sample attention sink patterns and value-state drains can be found in Fig. 8.19.

8.6.7 Redundancy of attention patterns

Numerous works have noted the prevalence of redundant attention patterns across attention heads, even from different layers [71, 72, 128, 129, 142], and attention sinks make up part of those patterns. In Fig. 8.20, we observe the same phenomena in recent pretrained LLMs. We plot attention matrices from the last 8 layers of five models. The chosen head indices are

simply ordered sequentially. Not only are attention patterns similar among heads of every model, different models have different sink token behaviors. Llama-2-7b-hf uses two sinks and divides weight between them. Llama-3.1-8B uses the first token as a sink. OLMo-2-1124-7B, like Llama-2-7b-hf, uses two sink tokens, but the second intermediate sink is at a different position than that of Llama-2-7b-hf. Qwen2.5-7B tends to use the *second* token as a sink, while Qwen2.5-14B uses the first token. We exclude Llama-3.2-3B from Fig. 8.20 because it presents similar attention patterns to Llama-3.1-8B.

For every model in Fig. 8.20, we also show the top 2 principal components across all attention matrices (of which there are $N_{\text{layer}} \times N_{\text{heads}}$). It is surprising to see a few principal components capturing more than half of the observed variance of attention matrices, for all models.

The input to all models in Fig. 8.20 is from MMLU's `high_school_computer_science`, dev split, index 2. The input text is in the same format used by `lm-evaluation-harness` [138]:

```
The following are multiple choice questions (with answers) about
high school computer science.


What is the output of "abc"[::-1] in Python 3?
A. Error
B. abc
C. cba
D. c
```

`Answer:`

8.6.8 On lm-evaluation-harness evaluations

For reliable and complete evaluations, use batch_size='auto' in lm-evaluation-harness. It not only finds an optimal batch size for your hardware to prevent memory errors but also mitigates the issue where an entire batch may be skipped because one sample cannot fit on the GPU.

8.6.9 PyTorch Implementation of Avg Head Output Norm (LN) and Avg Weight to First Token

We provide sample PyTorch code [143], for Avg Head Output Norm (LN) and Avg Weight to First Token, that can be integrated into a self-attention module's forward pass. When using `lm-evaluation-harness` [138] to evaluate pretrained models, additional steps must be taken to ignore padding tokens during log likelihood evaluations (on MC datasets).

All score functions are implemented within the self-attention module and take as input the following tensors: attention weights of size $(B, N_{\text{head}}, S, S)$, value states of size $(B, N_{\text{head}}, S, d_v)$, and a float threshold. B denotes the batch size, S denotes the sequence length, N_{head} denotes the number of attention heads, and d_v is the dimension of the value states. As output, both implementations return a boolean mask of dormant heads of size (B, N_{head}) and attention outputs of size $(B, N_{\text{head}}, S, d_v)$. In the dormant mask, an entry is True if the head is declared dormant, and False otherwise. Note that models that use grouped-query attention (GQA) [144] or multi-query attention (MQA) [145] still construct tensors of these sizes, so the specific kind of multi-head attention is not relevant.

Listing 8.1: Avg Weight to First Token in PyTorch

```python
def awft_dormant_mask(attn_weights, value_states, threshold):
    avg_weight = attn_weights.mean(dim=-2)             # (B, N_head, S)
    first_token_avg_weight = avg_weight[:,:,0]         # (B, N_head)
    dormant_mask = first_token_avg_weight > threshold # (B, N_head)

    # Model intervention: set dormant head outputs to zero
    attn_output = torch.matmul(attn_weights, value_states)
    attn_output[dormant_mask] = 0
    return attn_output, dormant_mask
```

Listing 8.2: Avg Head Output Norm (LN) in PyTorch, following Eq. (8.1)

```python
def ahon_ln_dormant_mask(attn_weights, value_states, threshold):
    attn_output = torch.matmul(attn_weights, value_states)
    norm_per_token = attn_output.norm(dim=-1)        # (B, N_head, S)
    avg_norm_per_head = norm_per_token.mean(dim=-1) # (B, N_head)

    # compute average across all heads in layer
    layer_context = avg_norm_per_head.mean(dim=1) # (B,)
    rel_avg_norm_per_head = (avg_norm_per_head / layer_context[:,
       None]) # (B, N_head)
    dormant_mask = rel_avg_norm_per_head < threshold # (B, N_head)

    # Model intervention: set dormant head outputs to zero
```

```
    attn_output[dormant_mask] = 0
    return attn_output, dormant_mask
```

8.6.10 Additional model and dataset information

Model details. Model inference is done in the original data type of the saved weights using `AutoModelForCausalLM.from_pretrained(...,torch_dtype="auto")`, except for OLMo-2 models, where we use float16. All models are downloaded using Hugging Face (HF) `transformers` [131]. Parameter counts and release dates are shown in Tab. 8.6.

Model Name	Params	Heads Per Layer	Num Layers	Total Heads	Release Date
Llama-3.1 8B	8.03B	32	32	1024	Jul 2024
Llama-3.2-3B	3.21B	24	28	672	Sep 2024
OLMo-2-1124-7B	7.3B	32	32	1024	Nov 2024
Qwen2.5-7B	7.62B	28	28	784	Feb 2025
Qwen2.5-14B	14.8B	40	48	1920	Feb 2025

Table 8.6: Additional model information of pretrained LLMs used in this work.

Dataset details. The test split of the MMLU benchmark contains 14,042 multiple-choice (MC) questions spanning 57 tasks including mathematics, US history, computer science, law, and more. Each question has 4 answer choices.

We also use PIQA [136] and WinoGrande [137], but only in Sec. 8.6.1 to reduce clutter in the main body. PIQA is a MC benchmark dataset of physical commonsense questions where each question has 2 answer choices. WinoGrande is a MC benchmark dataset of pronoun resolution problems where each problem has 2 answer choices.

8.6.11 Practical Implications and Limitations

Architectures. We think there is evidence that different problems require different fractions of model parameters to solve [146, 147]. Our experiments provide concrete evidence that this occurs in self-attention, and we think sparsity can be beneficial for designing architectures like MoEs but for attention modules. We may not need to have all heads in memory if only a subset will be used. A router could potentially be developed to select the appropriate heads. Our work provides necessary background on inactive attention heads so that future work can develop new architectures for this problem.

Efficiency. It should be possible to dynamically prune different heads on-the-fly whenever we identify a head as inactive. But it is unknown whether the overhead in identifying these heads (by computing scores) will make inference so slow that the compute savings are not worth it. In particular, our best score function (AHON (LN)) has the issue that it requires computing the head output itself, making it impossible to save compute when "skipping" a head. However, future work may discover a more proper score function that would allow for compute saving. For example, [147] showed that image classifiers could be dynamically pruned at runtime, for each individual input. Another potential direction is KV cache compression: specifically, not storing K/V for heads that are inactive. Our work also opens up new directions for investigation like finetuning inactive heads.

Limitations. We focus on analyzing a single forward pass because that is how log-likelihood evaluations of MMLU, PIQA, WinoGrande and other MC benchmarks are performed. Analyzing generation is complex because zeroing out a single incorrectly identified head can ruin the rest

of the generated sequence (*i.e.* more opportunities for failure). For example, if we zero out too many heads, we could sample a token representing a different language. Analyzing generation is a natural next step, as it is used for more practical use-cases.

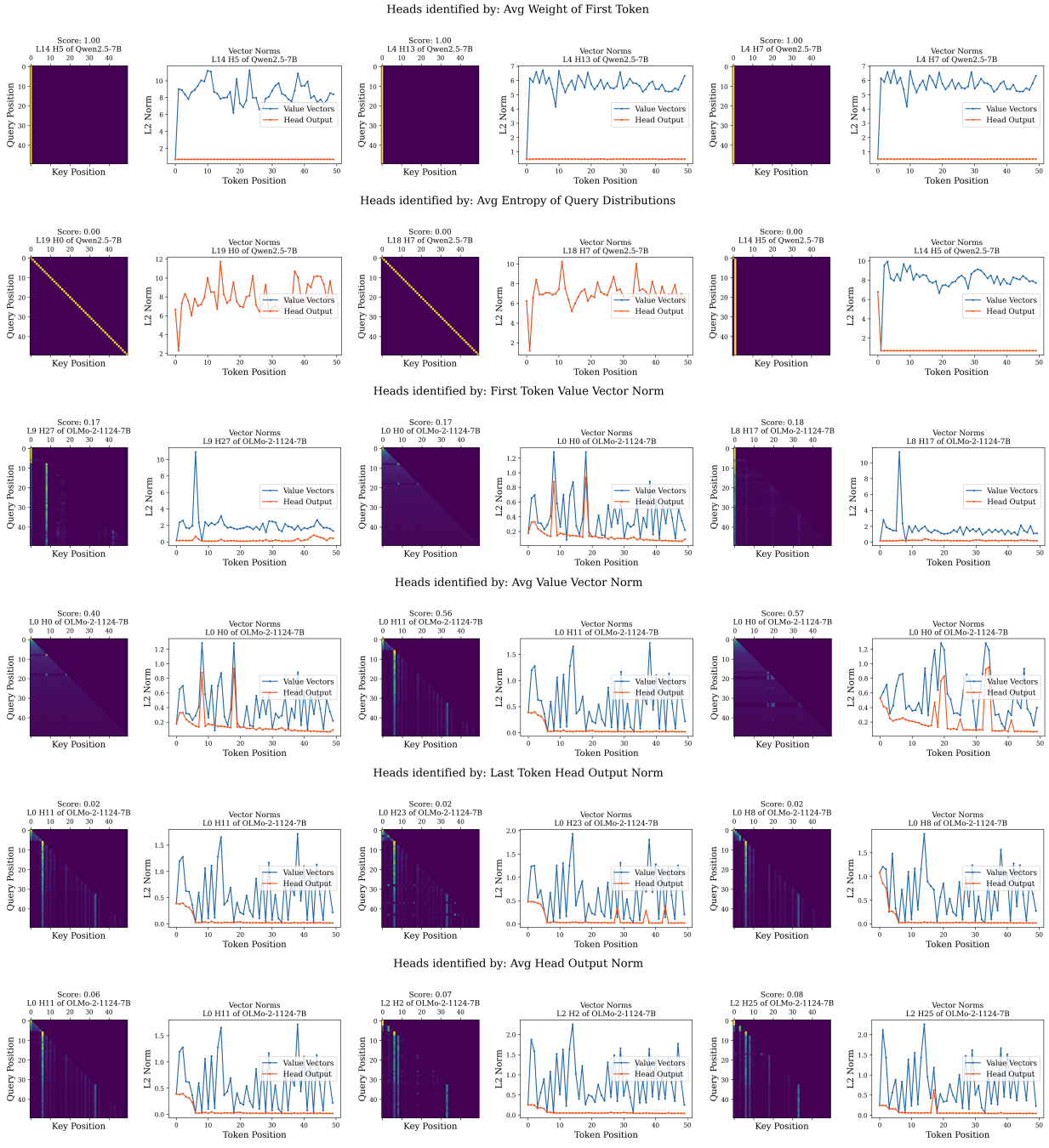


Figure 8.8: **Sample Attention Heads for 6 Unnormalized Score Functions.** In each row, we plot 3 sample attention heads for each score function in Tab. 8.1. To the right of each attention weights matrix, we plot the ℓ_2-norm of Value Vectors and Head Outputs. We also include the layer index and head index of each attention head above each plot. Lower scores are better for all score functions other than AWFT. Note the variety of heads that are identified by each score function.

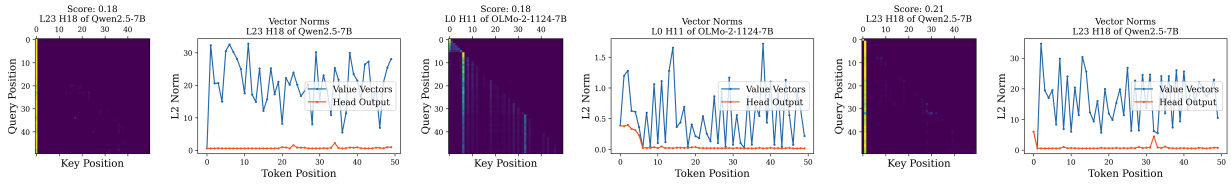


Figure 8.9: **Sample Attention Heads using Avg Head Output Norm (LN).** We plot 3 sample attention heads for AHON (LN), the best score function by AUC (See Fig. 8.3). To the right of each attention weights matrix, we plot the ℓ_2-norm of Value Vectors and Head Outputs. Note that the first head has a first token sink, the second head has two sinks, and the third head has a second token sink. AHON (LN) is able to capture all three distinct heads.

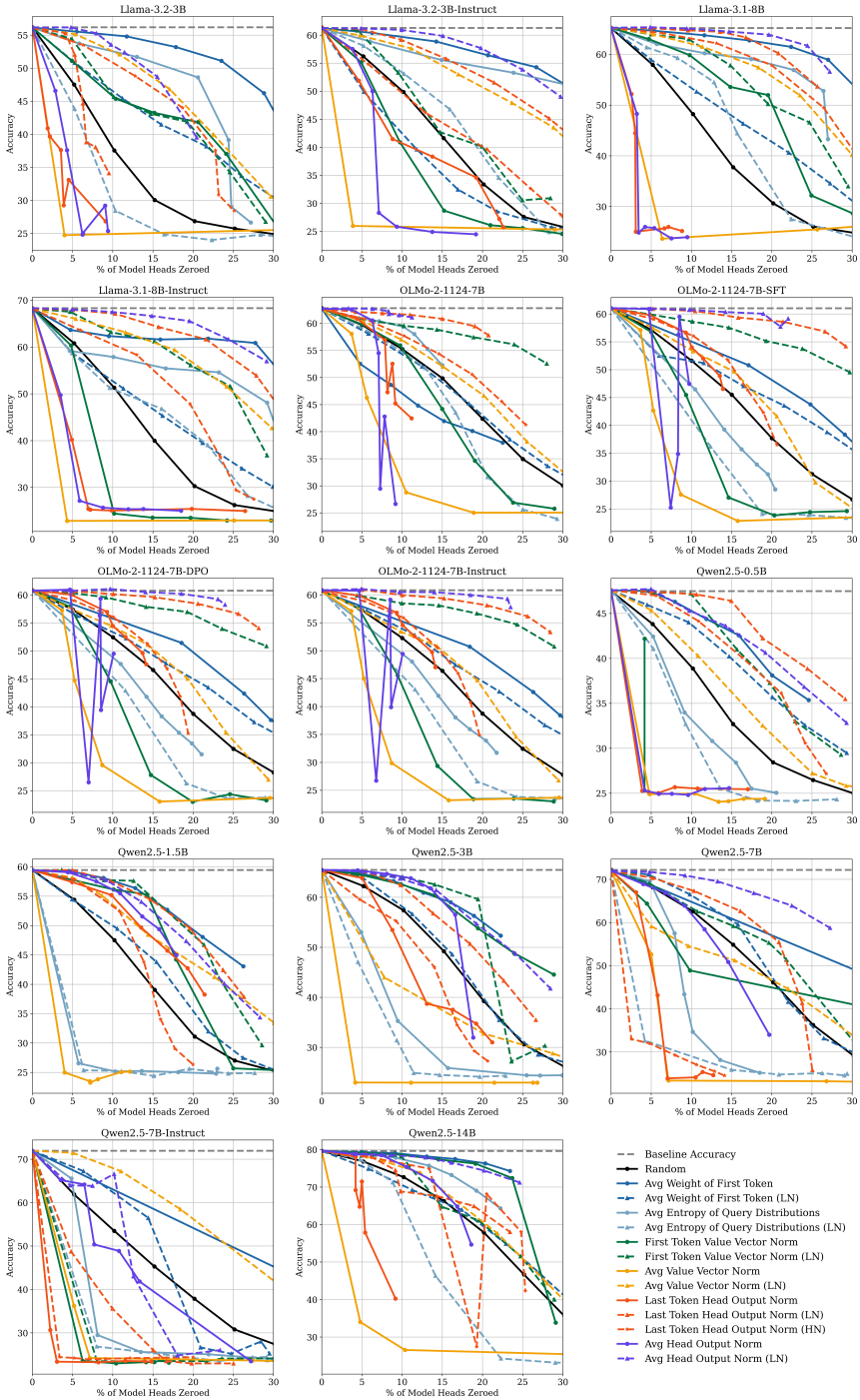


Figure 8.10: **Complete MMLU Results of Sec. 8.4.3.** For 14 models, we consider 13 score functions, and evaluate each at 7 distinct thresholds τ for each.

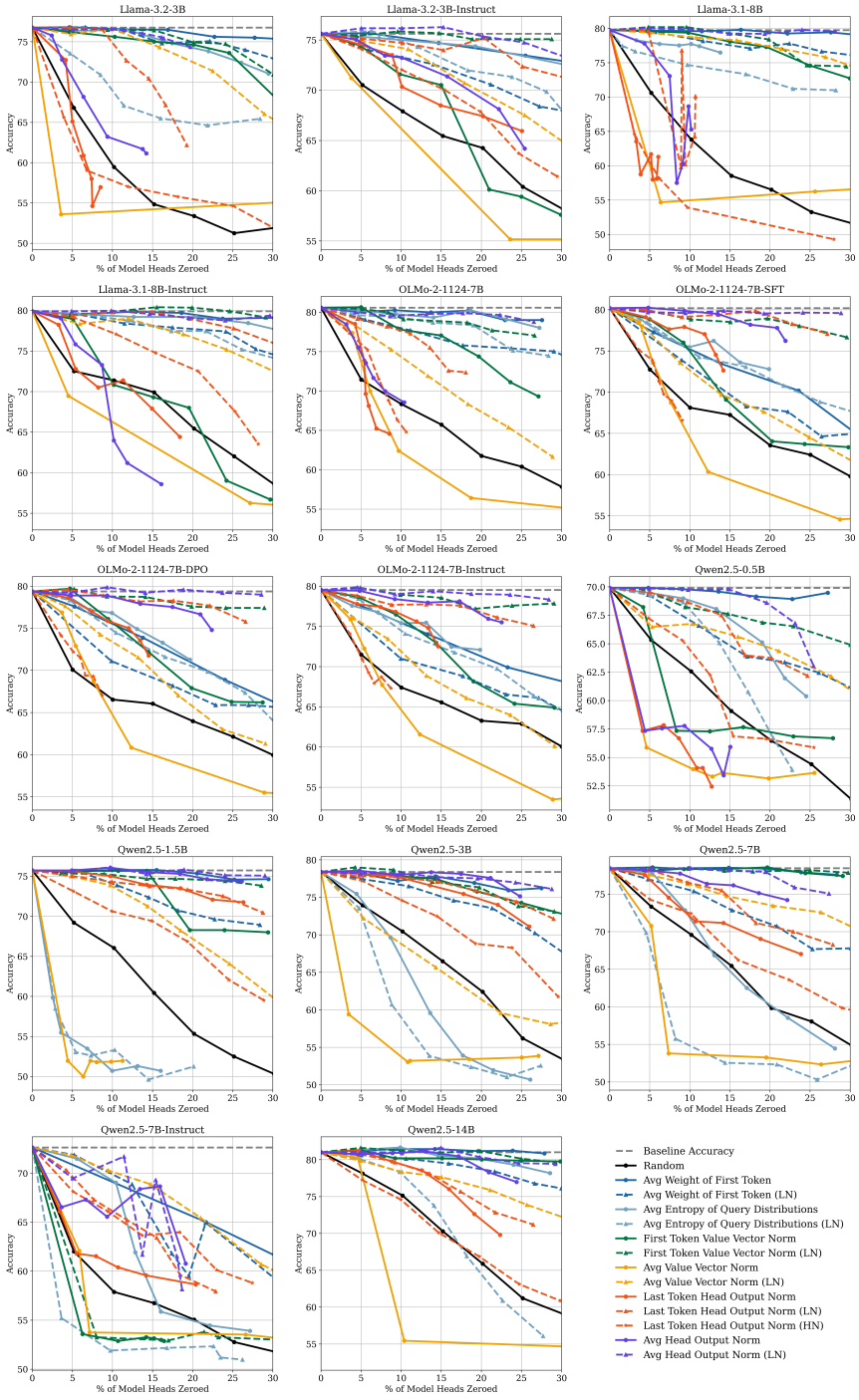


Figure 8.11: **Additional PIQA Results for Sec. 8.4.3.** For 14 models, we consider 13 score functions, and evaluate each at 7 distinct thresholds τ for each.

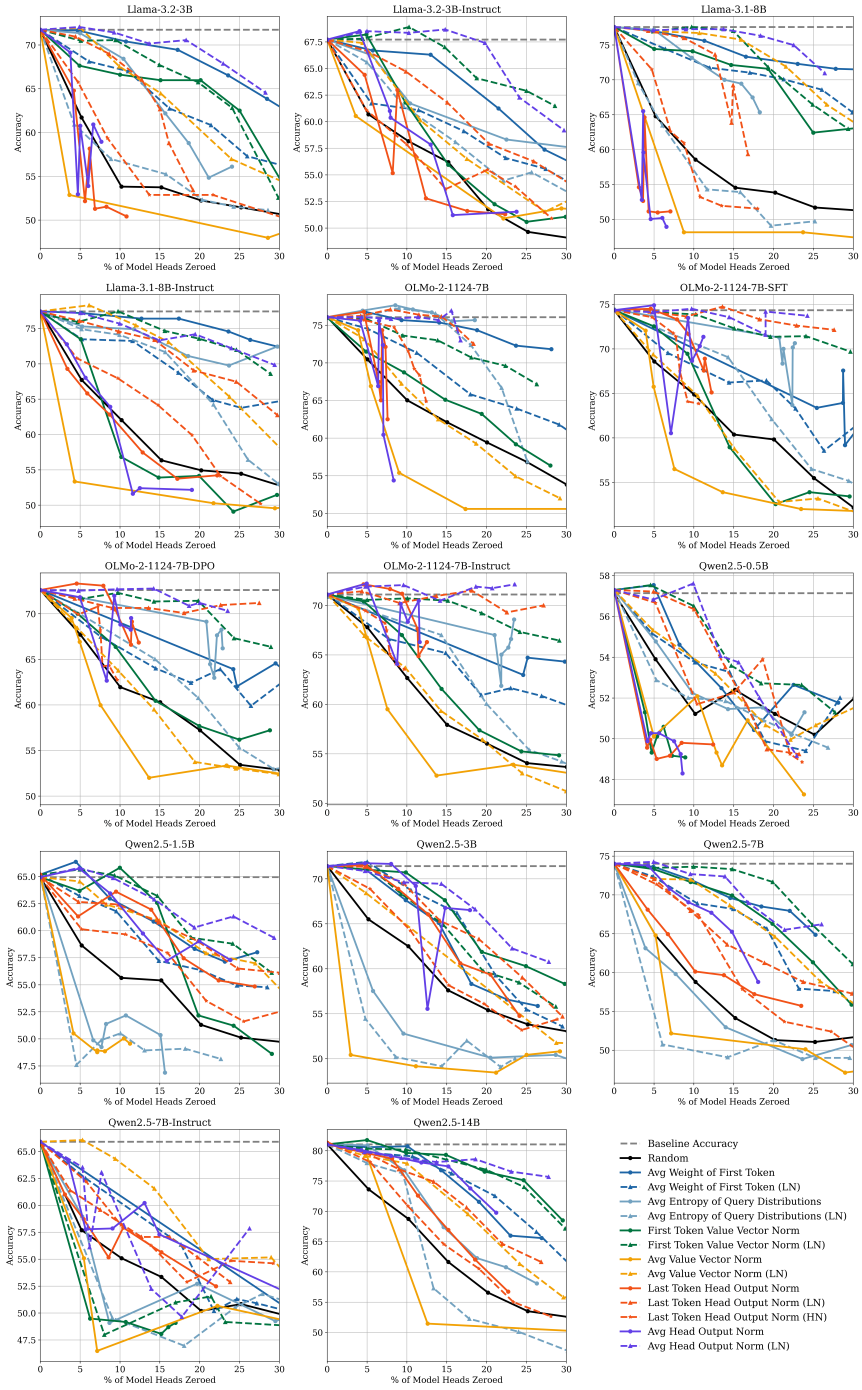


Figure 8.12: **Additional WinoGrande Results for Sec. 8.4.3.** For 14 models, we consider 13 score functions, and evaluate each at 7 distinct thresholds τ for each.

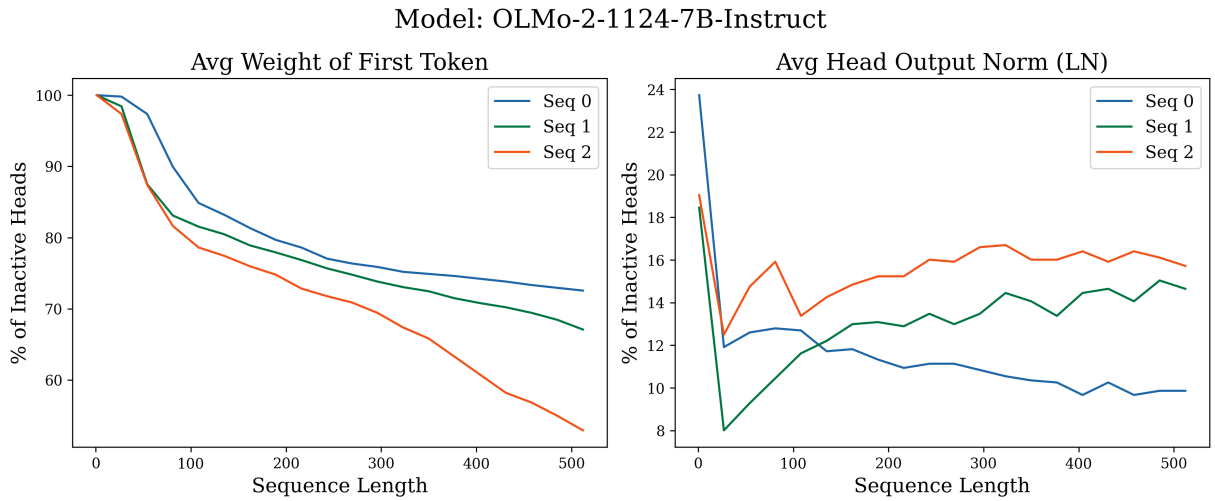


Figure 8.13: **Inactive Heads as Sequence Length Changes.** For OLMo-2-1124-7B-Instruct, we consider 3 FineWeb-Edu sequences, and evaluate the percent of inactive heads in the model as the sequence length increases. We compare AWFT ($\tau = 0.077$) and AHON (LN) ($\tau = 0.457$) score functions.

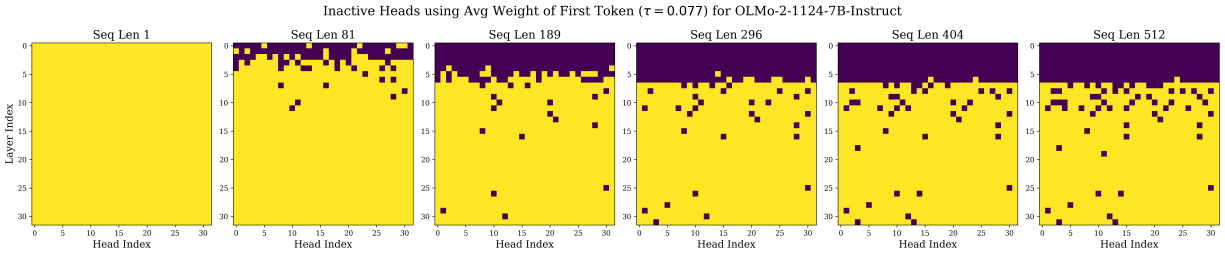


Figure 8.14: **Inactive Heads as Sequence Length Changes.** For OLMo-2-1124-7B-Instruct, we plot an inactive head mask for increasing sequence lengths of "Seq 0" from Fig. 8.13. Yellow indicates an inactive head, as identified by AWFT score function.

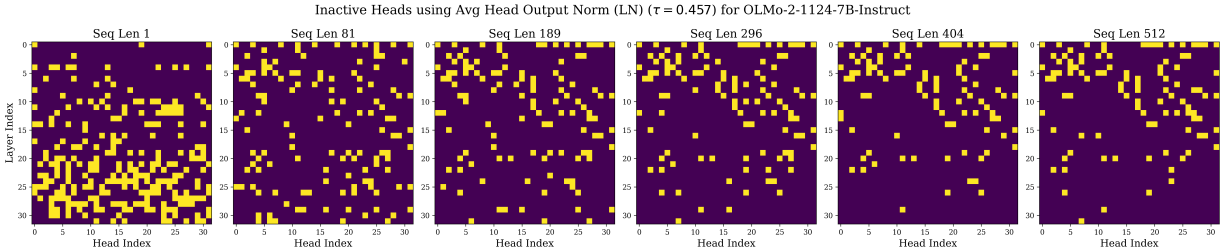


Figure 8.15: **Inactive Heads as Sequence Length Changes.** For OLMo-2-1124-7B-Instruct, we plot an inactive head mask for increasing sequence lengths of "Seq 0" from Fig. 8.13. Yellow indicates an inactive head, as identified by AHON (LN) score function.

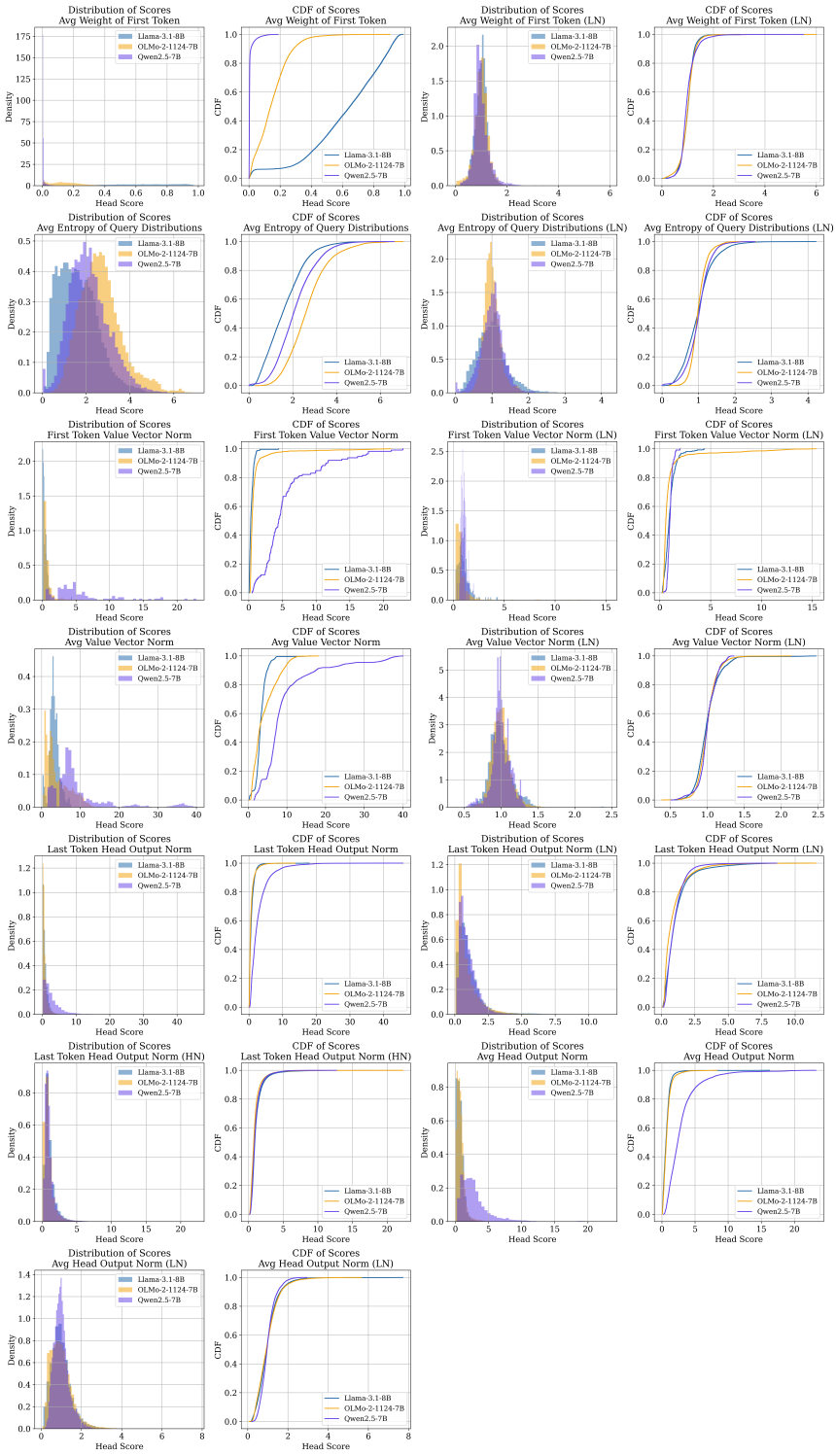


Figure 8.16: **Head Score Distributions on MMLU.** For 13 score functions, we plot a density distribution and a CDF of head scores. We only show scores for Llama-3.1-8B, OLMo-2-1124-7B, and Qwen2.5-7B for brevity.

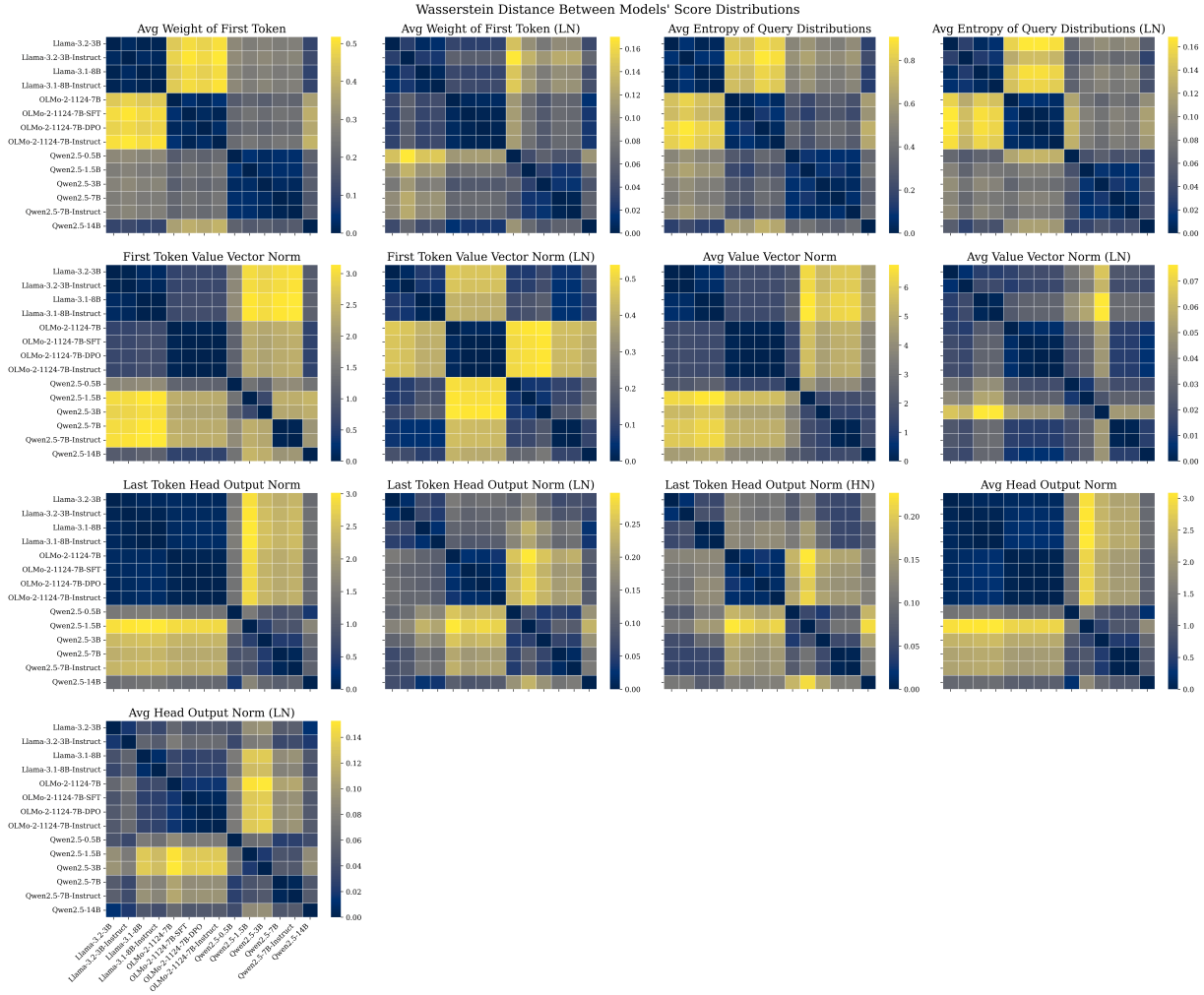


Figure 8.17: **Wasserstein Distance of Head Score Distributions between Models.** For each of 13 score functions, we plot a matrix of Wasserstein distances between score distributions of different models.

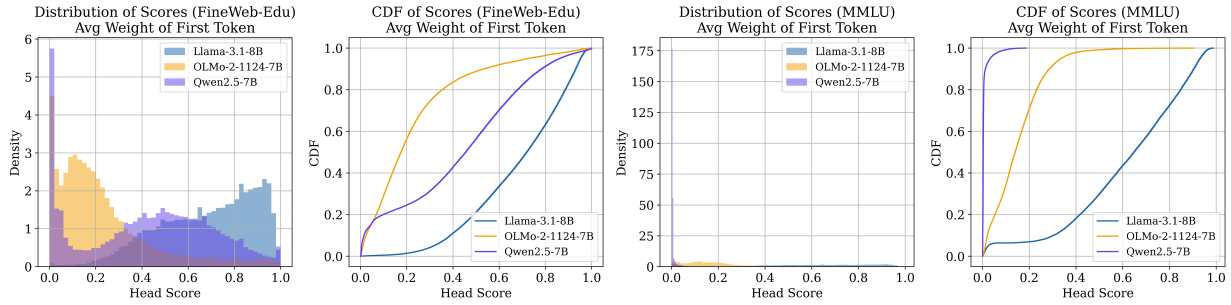


Figure 8.18: **Distribution of Avg Weight of First Token Scores changes when the input distribution changes.** Using the same 3 models, we find that the distribution of Avg Weight of First Token scores varies significantly when using input sequences from MMLU (right) instead of FineWeb-Edu (left).

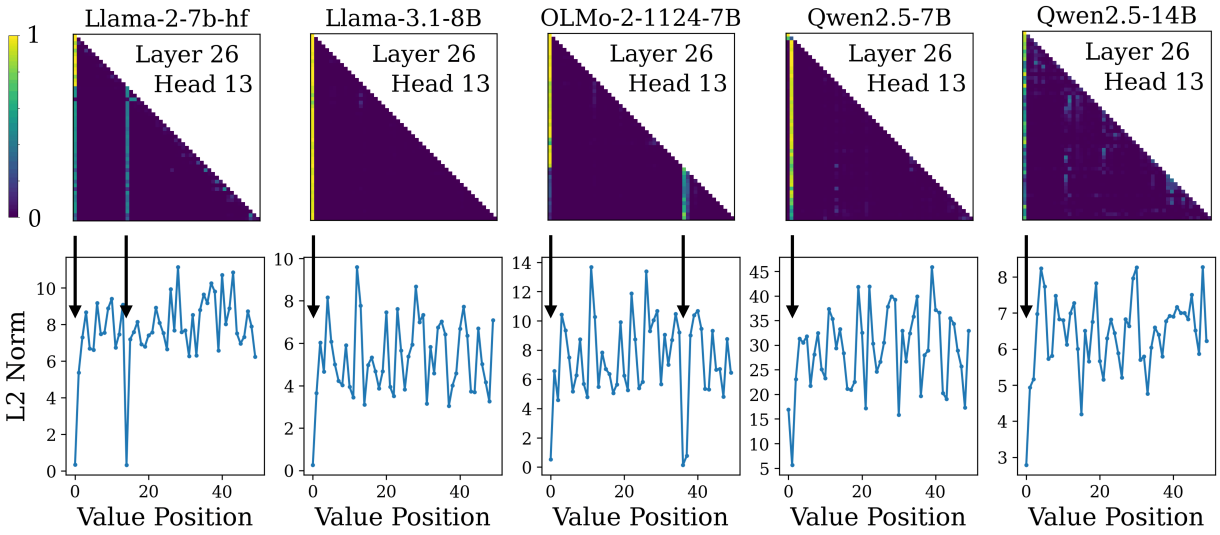


Figure 8.19: **Attention sink tokens have the smallest value vector norms**. For different models, we input the same question from MMLU. In the first row, we plot the attention weights for an arbitrary head (Layer 26, Head 13) across all models. In the second row, we plot the ℓ_2-norm of the value vector at every position.

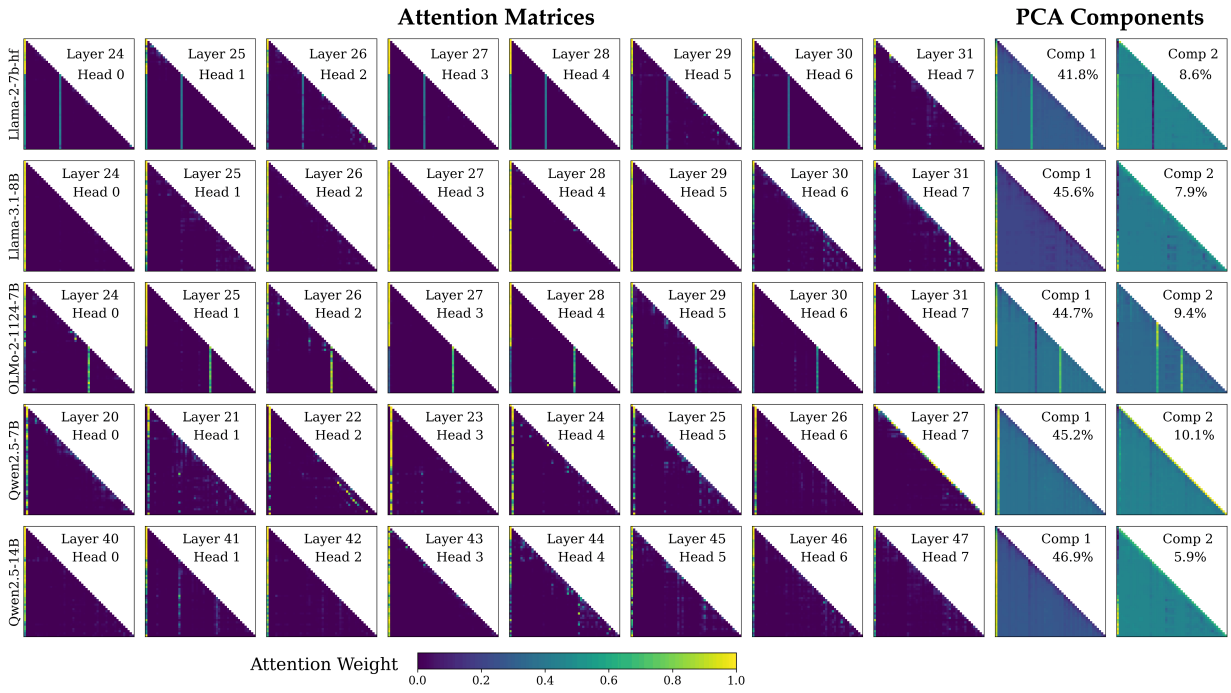


Figure 8.20: **PCA of Attention Matrices**. Attention patterns from different heads appear homogeneous and dominated by attention sinks. We input the same multiple-choice question from MMLU into Llama-2-7b-hf, Llama-3.1-8B, OLMo-2-1124-7B, Qwen2.5-7B, and Qwen2.5-14B then visualize the attention matrices. Each model row displays a sample of eight attention matrices from the last eight layers, followed by the top-2 principal components of all attention matrices, with the explained variance of each component displayed. The top-2 principal components display bright columns of attention sinks while capturing $\geq 50\%$ of the variance. "L26 H2" denotes the attention head at index 2 of layer 26.

Chapter 9: Conclusion

For a machine learning model to generalize, it must perform well on data not seen during training. It must extract patterns that extend beyond samples from the training set. To ensure this, common practice involves randomly partitioning data into a training set and a test set, so that the test set provides an estimate of the model's ability to generalize to unseen data. This relies on the i.i.d. assumption: that data is independent and identically distributed.

Our work on data poisoning relies on exploiting this very assumption. To create unlearnable data, we purposefully violate the i.i.d assumption: we perturb the training set with imperceptible perturbations that the test set does not contain. In Chapter 3, we explored dozens of patterns to understand whether ResNets are biased toward a particular kind of pattern. We find that *consistent* patterns that most clearly correlate with the label are best for poisoning. Among all the patterns we considered, synthetic "Regions" patterns take the fewest number of epochs for the network to learn the correlation.

Aiming to construct a perturbation that is most easily learned, in Chapter 4 we reached for autoregressive perturbations. All autoregressive signals have a corresponding convolutional filter that produces a zero response. This means that, without any training, we can construct a kernel capable of identifying any autoregressive signal. Empirically, we find that our autoregressive perturbations withstand attacks for utilizing the data while being imperceptible, model-free,

backpropagation-free, and reusable on different datasets.

In Chapter 5, we begin to question whether unlearnable datasets truly make data useless. Do unlearnable datasets hold up the promise of data protection. It turns out that corrupted unlearnable data sets still contain useful information. Through the Orthogonal Projection method, we show that when perturbations are applied class-wise, they occupy a low-dimensional subspace that can be projected away, and that models trained on the projected data still generalize.

Continuing the trend of breaking unbounded unlearnable data sets, in Chapter 6 we focus on learning from CUDA unlearnable datasets. The images in CUDA are heavily blurred, and the "perturbation" is unbounded. We develop random sharpening kernels that effectively recover enough information from CUDA datasets such that we achieve test accuracies comparable to adversarial training.

Liu et al. [103]'s demonstration that JPEG compression neutralizes unlearnable dataset perturbations led us to explore whether JPEG compression defeats protections against AI editing in Chapter 7. We show that JPEG compression ruins the imperceptible perturbation that protects images from AI editing. An image "protected" against AI-editing by a diffusion model can be made editable by simply JPEG-compressing the protected image.

Finally, in Chapter 8 we study pretrained language models, the transformer decoders capable of dozens of NLP tasks. Through a careful analysis of the self-attention module and attention weights, we are able to better pinpoint inactive attention heads, components of the models that are not needed during each forward pass. We discover a potential for improved efficiency by better understanding and quantifying the extent to which all attention hands are necessary. Previous methods estimated that less than 7% of heads were inactive, but our new score functions find that over 12% of heads in pretrained language models are not required in every forward pass. Better

understanding attention in transformers allowed us to measure and exploit inactive computation during inference.

Taken as a whole, this dissertation provides evidence that understanding what and how neural networks learn is essential. Only after understanding simplicity bias did we control it and develop perturbations for unlearnable datasets. Only after studying attention heads in pretrained transformers were we able to identify and measure inactive heads in modern language models. Understanding what and how neural networks learn allows us to better control them during both training and inference for a variety of purposes. We have only begun scratching the surface regarding how neural networks are able to both generalize and suffer from shortcut learning, but I am confident that by understanding these systems better, we will be able to improve them.

For practitioners, my recommendations are simple: spend ample amounts of time analyzing the statistical properties of your data and, after training, carefully audit how your neural network works. What one expects a neural network to have learned is not always what it actually learned. When a neural net behaves differently than expected, understanding why either advances our scientific understanding of deep learning or reveals specific opportunities for how to improve it.

9.1 Future Work

9.1.1 Sharing Data Using Latent Codes

While it is possible to optimize perturbations that cause overfitting in neural networks, the success of these modified datasets in protecting data remains elusive. As with most attacker-defender scenarios, one next step might be to integrate knowledge of the attacker in crafting the next defense. This would mean crafting perturbations that are robust to JPEG compression and

blurs, among other transformations. Still, it is intractable to enumerate all possible transformations a web scraper might execute on data prior to utilizing it. Rather than focus on how to optimize protections, we could instead rethink how we share the data in the first place.

The unlearnable dataset premise is one where a data owner wants to allow others to view data, but not utilize the data. If modifications to the original data retain too much information, as we show in Chapter 5, one approach could be to use the latent codes from a VQ-VAE [148] to "tokenize" the image prior to sharing. Because VQ-VAEs inherently use an information bottleneck, data viewers will be unable to reconstruct the original image while still being able to view a similar image. Additionally, the complexity of the perturbation (or reconstruction noise) would be much harder to characterize, as it may involve changes to image content and layout. Relative to the original image, the perturbation would be very different from the additive patterns we explored in Chapters 3 and 4. While the CUDA dataset of Chapter 6 did not employ additive perturbations, one could argue the perturbation still maintained image content and layout.

9.1.2 Transformer Efficiency

The results in Chapter 8 demonstrate that not all computation in transformer neural networks is necessary. The more we understand about how computation works, the more we can optimize. For instance, because attention sinks are a consistent learned behavior across language models with softmax attention, making it easier to acquire this behavior could yield training time efficiencies because it would no longer need to be learned from data. Indeed, the recent GPT-OSS model implements a per-head "sink parameter" that allows every attention head to attenuate its entire attention output by a learned amount [149]. Other architectural options include clipped softmax

and gated attention [150].

While architectural improvements have the potential to yield training time efficiencies, improving datasets have already improved training efficiency, and there is likely more to be gained [151]. One exciting direction would be to study the circuits used by transformer language models [69], then construct data tailored for learning those circuits. One could focus on inference in one particular task and optimize data for that domain.

Inference time improvements are also possible. But more work is needed to predict inactive attention heads without having to calculate attention head outputs. One approach would be to train lightweight, head-specific classifiers that operate only on hidden states and predict whether a head is inactive. Even then, one would need to concatenate head outputs (although some would be zero), and execute an output projection. Another idea would be to scale attention heads while keeping inference speed fixed by developing "mixture of attention heads," an analog to mixture of experts [152]. An attention head router could determine which subsets of attention heads to compute with, allowing attention parameters to increase alongside feedforward parameters. Across dataset security and transformer inference, a deeper understanding of what neural networks learn enables the development of more effective algorithms, more robust analyses, and new opportunities for improving machine learning systems.

Bibliography

[1] Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, 2020.

[2] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[3] John R Zech, Marcus A Badgeley, Manway Liu, Anthony B Costa, Joseph J Titano, and Eric Karl Oermann. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. *PLoS medicine*, 15(11): e1002683, 2018.

[4] Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2021–2031, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1215. URL https://aclanthology.org/D17-1215/.

[5] Eric Wallace, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. Universal adversarial triggers for attacking and analyzing NLP. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2153–2162, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1221. URL https://aclanthology.org/D19-1221/.

[6] Stephanie Lin, Jacob Hilton, and Owain Evans. TruthfulQA: Measuring how models mimic human falsehoods. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio, editors, *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3214–3252, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.229. URL https://aclanthology.org/2022.acl-long.229/.

[7] R. Thomas McCoy, Ellie Pavlick, and Tal Linzen. Right for the wrong reasons: Diagnosing syntactic heuristics in natural language inference. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3428–3448, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1334. URL https://aclanthology.org/P19-1334/.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[10] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *International Conference on Machine Learning*, pages 8821–8831. PMLR, 2021.

[11] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125*, 2022.

[12] Abeba Birhane and Vinay Uday Prabhu. Large image datasets: A pyrrhic win for computer vision? In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1536–1546. IEEE, 2021.

[13] Kashmir Hill. The secretive company that might end privacy as we know it, Jan 2020. URL `https://www.nytimes.com/2020/01/18/technology/clearview-privacy-facial-recognition.html`.

[14] Micah Goldblum, Dimitris Tsipras, Chulin Xie, Xinyun Chen, Avi Schwarzschild, Dawn Song, Aleksander Madry, Bo Li, and Tom Goldstein. Dataset security for machine learning: Data poisoning, backdoor attacks, and defenses. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

[15] N. Carlini, M. Jagielski, C. Choquette-Choo, D. Paleka, W. Pearce, H. Anderson, A. Terzis, K. Thomas, and F. Tramèr. Poisoning web-scale training datasets is practical. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 176–176, Los Alamitos, CA, USA, may 2024. IEEE Computer Society. doi: 10.1109/SP54263.2024.00094. URL `https://doi.ieeecomputersociety.org/10.1109/SP54263.2024.00094`.

[16] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. Laion-400m: Open dataset of clip-filtered 400 million image-text pairs. *arXiv preprint arXiv:2111.02114*, 2021.

[17] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 1467–1474, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

[18] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6572.

[19] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*, pages 99–112. Chapman and Hall/CRC, 2018.

[20] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[21] Nicholas Carlini and David Wagner. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, May 2017. doi: 10.1109/SP.2017.49. ISSN: 2375-1207.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJzIBfZAb.

[23] Xinyun Chen, Chang Liu, Bo Li, Kimberly Lu, and Dawn Song. Targeted backdoor attacks on deep learning systems using data poisoning. *arXiv preprint arXiv:1712.05526*, 2017.

[24] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. Badnets: Identifying vulnerabilities in the machine learning model supply chain. *arXiv preprint arXiv:1708.06733*, 2017.

[25] Ali Shafahi, W Ronny Huang, Mahyar Najibi, Octavian Suciu, Christoph Studer, Tudor Dumitras, and Tom Goldstein. Poison frogs! targeted clean-label poisoning attacks on neural networks. *Advances in Neural Information Processing Systems*, 31, 2018.

[26] Chen Zhu, W Ronny Huang, Hengduo Li, Gavin Taylor, Christoph Studer, and Tom Goldstein. Transferable clean-label poisoning attacks on deep neural nets. In *International Conference on Machine Learning*, pages 7614–7623. PMLR, 2019.

[27] Marco Barreno, Blaine Nelson, Anthony D Joseph, and J Doug Tygar. The security of machine learning. *Machine Learning*, 81(2):121–148, 2010.

[28] Hanxun Huang, Xingjun Ma, Sarah Monazam Erfani, James Bailey, and Yisen Wang. Unlearnable examples: Making personal data unexploitable. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=iAmZUo0DxC0.

[29] Liam Fowl, Micah Goldblum, Ping-yeh Chiang, Jonas Geiping, Wojciech Czaja, and Tom Goldstein. Adversarial examples make strong poisons. In M. Ranzato, A. Beygelzimer,

Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 30339–30351. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper/2021/file/fe87435d12ef7642af67d9bc82a8b3cd-Paper.pdf.

[30] Liam Fowl, Ping-yeh Chiang, Micah Goldblum, Jonas Geiping, Arpit Bansal, Wojtek Czaja, and Tom Goldstein. Preventing unauthorized use of proprietary data: Poisoning for secure dataset release. *arXiv preprint arXiv:2103.02683*, 2021.

[31] Weiqi Peng and Jinghui Chen. Learnability lock: Authorized learnability control through adversarial invertible transformations. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=6VpeS27viTq.

[32] Pedro Sandoval-Segura, Vasu Singla, Jonas Geiping, Micah Goldblum, Tom Goldstein, and David Jacobs. Autoregressive perturbations for data poisoning. *Advances in Neural Information Processing Systems*, 35:27374–27386, 2022.

[33] Da Yu, Huishuai Zhang, Wei Chen, Jian Yin, and Tie-Yan Liu. Availability attacks create shortcuts. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 2367–2376, 2022.

[34] Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12230–12240. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/yuan21b.html.

[35] Lue Tao, Lei Feng, Jinfeng Yi, Sheng-Jun Huang, and Songcan Chen. Better safe than sorry: Preventing delusive adversaries with adversarial training. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 16209–16225. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/8726bb30dc7ce15023daa8ff8402bcfd-Paper.pdf.

[36] Vinu Sankar Sadasivan, Mahdi Soltanolkotabi, and Soheil Feizi. Cuda: Convolution-based unlearnable datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3862–3871, 2023.

[37] Pedro Sandoval-Segura, Vasu Singla, Jonas Geiping, Micah Goldblum, and Tom Goldstein. What can we learn from unlearnable datasets? *Advances in Neural Information Processing Systems*, 36, 2024.

[38] Shutong Wu, Sizhe Chen, Cihang Xie, and Xiaolin Huang. One-pixel shortcut: On the learning preference of deep neural networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=p7G8t5FVn2h.

[39] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[40] Juncheng Shen, Xiaolei Zhu, and De Ma. Tensorclog: An imperceptible poisoning attack on deep neural network applications. *IEEE Access*, 7:41498–41506, 2019.

[41] Ji Feng, Qi-Zhi Cai, and Zhi-Hua Zhou. Learning to confuse: generating training time adversarial data with auto-encoder. *Advances in Neural Information Processing Systems*, 32, 2019.

[42] Fu Shaopeng, He Fengxiang, Liu Yang, Shen Li, and Tao Dacheng. Robust unlearnable examples: Protecting data privacy against adversarial learning. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=baUQQPwQiAg.

[43] Zhirui Wang, Yifei Wang, and Yisen Wang. Fooling adversarial training with inducing noise. *arXiv preprint arXiv:2111.10130*, 2021.

[44] Pedro Sandoval-Segura, Vasu Singla, Liam Fowl, Jonas Geiping, Micah Goldblum, David Jacobs, and Tom Goldstein. Poisons that are learned faster are more effective. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 198–205, 2022.

[45] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

[46] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032, 2019.

[47] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.

[48] Jonas Geiping, Liam Fowl, Gowthami Somepalli, Micah Goldblum, Michael Moeller, and Tom Goldstein. What doesn't kill you makes you robust (er): How to adversarially train against data poisoning. *arXiv preprint arXiv:2102.13624*, 2021.

[49] Eric Wallace, Tony Zhao, Shi Feng, and Sameer Singh. Concealed data poisoning attacks on nlp models. In *North American Chapter of the Association for Computational Linguistics*, 2021. URL https://api.semanticscholar.org/CorpusID:233230124.

[50] Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

[51] Pang Wei Koh, Shiori Sagawa, Henrik Marklund, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, Etienne David, Ian Stavness, Wei Guo, Berton Earnshaw, Imran Haque, Sara M Beery, Jure Leskovec, Anshul Kundaje, Emma Pierson, Sergey Levine, Chelsea Finn, and Percy Liang. Wilds: A benchmark of in-the-wild distribution shifts. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 5637–5664. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/koh21a.html.

[52] Harshay Shah, Kaustav Tamuly, Aditi Raghunathan, Prateek Jain, and Praneeth Netrapalli. The pitfalls of simplicity bias in neural networks. *Advances in Neural Information Processing Systems*, 33:9573–9585, 2020.

[53] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJz6tiCqYm.

[54] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bygh9j09KX.

[55] Andrei Barbu, David Mayo, Julian Alverio, William Luo, Christopher Wang, Dan Gutfreund, Josh Tenenbaum, and Boris Katz. Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models. *Advances in neural information processing systems*, 32, 2019.

[56] Zuxuan Wu, Ser-Nam Lim, Larry S Davis, and Tom Goldstein. Making an invisibility cloak: Real world adversarial attacks on object detectors. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part IV 16*, pages 1–17. Springer, 2020.

[57] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyxAb30cY7.

[58] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In *international conference on machine learning*, pages 1310–1320. PMLR, 2019.

[59] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[60] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[61] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[62] Ting Chen, Saurabh Saxena, Lala Li, David J Fleet, and Geoffrey Hinton. Pix2seq: A language modeling framework for object detection. *arXiv preprint arXiv:2109.10852*, 2021.

[63] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International conference on machine learning*, pages 28492–28518. PMLR, 2023.

[64] Oleg Atamanenko, Anna Chalova, Joseph Coombes, Nikki Cope, Phillip Dang, Zhifeng Deng, Jimmy Du, Michael Ermolenko, Feifan Fan, Yufei Feng, et al. Tts-1 technical report. *arXiv preprint arXiv:2507.21138*, 2025.

[65] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4195–4205, 2023.

[66] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. In *Proceedings of the 2019 ACL workshop BlackboxNLP: analyzing and interpreting neural networks for NLP*, pages 276–286, 2019.

[67] Oriane Siméoni, Gilles Puy, Huy V. Vo, Simon Roburin, Spyros Gidaris, Andrei Bursuc, Patrick Pérez, Renaud Marlet, and Jean Ponce. Localizing objects with self-supervised transformers and no labels. November 2021.

[68] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[69] Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Nova DasSarma, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 2021. https://transformer-circuits.pub/2021/framework/index.html.

[70] Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, et al. In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*, 2022.

[71] Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NG7sS51zVF.

[72] Tianyu Guo, Druv Pai, Yu Bai, Jiantao Jiao, Michael I. Jordan, and Song Mei. Active-dormant attention heads: Mechanistically demystifying extreme-token phenomena in llms, 2024. URL https://arxiv.org/abs/2410.13835.

[73] Mingjie Sun, Xinlei Chen, J Zico Kolter, and Zhuang Liu. Massive activations in large language models. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=F7aAhfitX6.

[74] Zhongzhi Yu, Zheng Wang, Yonggan Fu, Huihong Shi, Khalid Shaikh, and Yingyan Celine Lin. Unveiling and harnessing hidden attention sinks: Enhancing large language models without training through attention calibration. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=DLTjFFiuUJ.

[75] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[76] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[77] Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[78] Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025.

[79] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[80] Cassidy Laidlaw and Soheil Feizi. Functional adversarial attacks. *Advances in neural information processing systems*, 32, 2019.

[81] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HyydRMZC-.

[82] Nicholas Baker, Hongjing Lu, Gennady Erlikhman, and Philip J Kellman. Local features and global shape information in object classification by deep convolutional neural networks. *Vision research*, 172:46–61, 2020.

[83] Guillermo Ortiz-Jimenez, Apostolos Modas, Seyed-Mohsen Moosavi, and Pascal Frossard. Hold me tight! influence of discriminative features on deep network boundaries. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2935–2946. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/1ea97de85eb634d580161c603422437f-Paper.pdf.

[84] Chuan Guo, Jared S. Frank, and Kilian Q. Weinberger. Low frequency adversarial perturbation. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 1127–1137. PMLR, 22–25 Jul 2020. URL https://proceedings.mlr.press/v115/guo20a.html.

[85] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.1556.

[86] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[87] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, D. Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[88] Herman Wold. *A study in the analysis of stationary time series*. PhD thesis, Almqvist & Wiksell, 1938.

[89] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[90] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[91] Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223. JMLR Workshop and Conference Proceedings, 2011.

[92] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. *Advances in neural information processing systems*, 32, 2019.

[93] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[94] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[95] Evani Radiya-Dixit, Sanghyun Hong, Nicholas Carlini, and Florian Tramer. Data poisoning won't save you from facial recognition. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=B5XahNLmna.

[96] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115: 211–252, 2015.

[97] Polina Kirichenko, Pavel Izmailov, and Andrew Gordon Wilson. Last layer re-training is sufficient for robustness to spurious correlations. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=Zb6c8A-Fghk.

[98] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[99] Rui Wen, Zhengyu Zhao, Zhuoran Liu, Michael Backes, Tianhao Wang, and Yang Zhang. Is adversarial training really a silver bullet for mitigating data poisoning? In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=zKvm1ETDOq.

[100] Hadi M. Dolatabadi, Sarah Erfani, and Christopher Leckie. The devil's advocate: Shattering the illusion of unexploitable data using diffusion models, 2023.

[101] Tianrui Qin, Xitong Gao, Juanjuan Zhao, Kejiang Ye, and Cheng-Zhong Xu. Learning the unlearnable: Adversarial augmentations suppress unlearnable example attacks. *arXiv preprint arXiv:2303.15127*, 2023.

[102] Dohyun Kim and Pedro Sandoval-Segura. Learning from convolution-based unlearnable datasets. In *The Third Workshop on New Frontiers in Adversarial Machine Learning*, 2024. URL https://openreview.net/forum?id=6gIEsRF38O.

[103] Zhuoran Liu, Zhengyu Zhao, and Martha Larson. Image shortcut squeezing: Countering perturbative availability poisons with compression. In *International conference on machine learning*, pages 22473–22487. PMLR, 2023.

[104] Nilaksh Das, Madhuri Shanbhogue, Shang-Tse Chen, Fred Hohman, Li Chen, Michael E Kounavis, and Duen Horng Chau. Keeping the bad guys out: Protecting and vaccinating deep learning with jpeg compression. *arXiv preprint arXiv:1705.02900*, 2017.

[105] Haohan Wang, Xindi Wu, Zeyi Huang, and Eric P Xing. High-frequency component helps explain the generalization of convolutional neural networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8684–8694, 2020.

[106] Pedro Sandoval-Segura, Jonas Geiping, and Tom Goldstein. Jpeg compressed images can bypass protections against ai editing. *arXiv preprint arXiv:2304.02234*, 2023.

[107] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33:6840–6851, 2020.

[108] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[109] Hadi Salman, Alaa Khaddaj, Guillaume Leclerc, Andrew Ilyas, and Aleksander Madry. Raising the cost of malicious ai-powered image editing. *arXiv preprint arXiv:2302.06588*, 2023.

[110] Shawn Shan, Jenna Cryan, Emily Wenger, Haitao Zheng, Rana Hanocka, and Ben Y Zhao. Glaze: Protecting artists from style mimicry by text-to-image models. *arXiv preprint arXiv:2302.04222*, 2023.

[111] Usage statistics of jpeg for websites. URL `https://w3techs.com/technologies/details/im-jpeg`.

[112] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. In *International conference on machine learning*, pages 284–293. PMLR, 2018.

[113] Richard Shin and Dawn Song. Jpeg-resistant adversarial images. In *NIPS 2017 Workshop on Machine Learning and Computer Security*, volume 1, page 8, 2017.

[114] Nataniel Ruiz, Yuanzhen Li, Varun Jampani, Yael Pritch, Michael Rubinstein, and Kfir Aberman. Dreambooth: Fine tuning text-to-image diffusion models for subject-driven generation. 2022.

[115] Pedro Sandoval-Segura, Xijun Wang, Ashwinee Panda, Micah Goldblum, Ronen Basri, Tom Goldstein, and David Jacobs. Identifying and evaluating inactive heads in pretrained LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL `https://openreview.net/forum?id=dfYMjaiMG4`.

[116] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[117] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[118] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, et al. 2 olmo 2 furious. *arXiv preprint arXiv:2501.00656*, 2024.

[119] An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[120] Liang Chen, Haozhe Zhao, Tianyu Liu, Shuai Bai, Junyang Lin, Chang Zhou, and Baobao Chang. An image is worth 1/2 tokens after layer 2: Plug-and-play inference acceleration for large vision-language models. In *European Conference on Computer Vision*, pages 19–35. Springer, 2025.

[121] Xiangming Gu, Tianyu Pang, Chao Du, Qian Liu, Fengzhuo Zhang, Cunxiao Du, Ye Wang, and Min Lin. When attention sink emerges in language models: An empirical view. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=78Nn4QJTEN.

[122] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1580. URL https://aclanthology.org/P19-1580/.

[123] Paul Michel, Omer Levy, and Graham Neubig. Are sixteen heads really better than one? *Advances in neural information processing systems*, 32, 2019.

[124] Maximiliana Behnke and Kenneth Heafield. Losing heads in the lottery: Pruning transformer. In *The 2020 Conference on Empirical Methods in Natural Language Processing*, pages 2664–2674. Association for Computational Linguistics (ACL), 2020.

[125] Jiaoda Li, Ryan Cotterell, and Mrinmaya Sachan. Differentiable subset pruning of transformer heads. *Transactions of the Association for Computational Linguistics*, 9:1442–1459, 2021.

[126] Zhiyu Guo, Hidetaka Kamigaito, and Taro Watanabe. Attention score is not all you need for token importance indicator in KV cache reduction: Value also matters. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 21158–21166, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1178. URL https://aclanthology.org/2024.emnlp-main.1178/.

[127] Goro Kobayashi, Tatsuki Kuribayashi, Sho Yokoi, and Kentaro Inui. Attention is not only a weight: Analyzing transformers with vector norms. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7057–7075, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.574. URL https://aclanthology.org/2020.emnlp-main.574/.

[128] Zichang Liu, Aditya Desai, Fangshuo Liao, Weitao Wang, Victor Xie, Zhaozhuo Xu, Anastasios Kyrillidis, and Anshumali Shrivastava. Scissorhands: Exploiting the persistence of importance hypothesis for llm kv cache compression at test time. *Advances in Neural Information Processing Systems*, 36:52342–52364, 2023.

[129] Suyu Ge, Yunan Zhang, Liyuan Liu, Minjia Zhang, Jiawei Han, and Jianfeng Gao. Model tells you what to discard: Adaptive KV cache compression for LLMs. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=uNrFpDPMyo.

[130] Qi Sun, Marc Pickett, Aakash Kumar Nain, and Llion Jones. Transformer layers as painters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25219–25227, 2025.

[131] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

[132] Meta. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, Sep 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

[133] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[134] Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval, 5 2023.

[135] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

[136] Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439, 2020.

[137] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: an adversarial winograd schema challenge at scale. *Commun. ACM*, 64(9):99–106, August 2021. ISSN 0001-0782. doi: 10.1145/3474381. URL https://doi.org/10.1145/3474381.

[138] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 07 2024. URL https://zenodo.org/records/12608602.

[139] Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

[140] Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

[141] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[142] Yongyu Mu, Yuzhang Wu, Yuchun Fan, Chenglong Wang, Hengyu Li, Qiaozhi He, Murun Yang, Tong Xiao, and Jingbo Zhu. Cross-layer attention sharing for large language models, 2024. URL https://arxiv.org/abs/2408.01890.

[143] A Paszke. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[144] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*, 2023. URL https://openreview.net/forum?id=hmOwOZWzYE.

[145] Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

[146] David Raposo, Sam Ritter, Blake Richards, Timothy Lillicrap, Peter Conway Humphreys, and Adam Santoro. Mixture-of-depths: Dynamically allocating compute in transformer-based language models. *arXiv preprint arXiv:2404.02258*, 2024.

[147] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

[148] Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[149] OpenAI. gpt_oss/torch/model.py. https://github.com/openai/gpt-oss/blob/d21f34ec3c1ede813adfbd83f07d2ad2eec7ff02/gpt_oss/torch/model.py#L153, 2025. GitHub repository, commit d21f34ec3c1ede813adfbd83f07d2ad2eec7ff02, accessed July 21, 2026.

[150] Yelysei Bondarenko, Markus Nagel, and Tijmen Blankevoort. Quantizable transformers: Removing outliers by helping attention heads do nothing. *Advances in Neural Information Processing Systems*, 36:75067–75096, 2023.

[151] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

[152] Noam Shazeer, *Azalia Mirhoseini, *Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=B1ckMDqlg.