On Internal Interfaces in Machine Learning Systems

Jonathan Hayase

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2026

Reading Committee:

Sewoong Oh, Chair

Noah A Smith

Pang Wei Koh

Program Authorized to Offer Degree:

Computer Science & Engineering

University of Washington

Abstract

On Internal Interfaces in Machine Learning Systems

Jonathan Hayase

Chair of the Supervisory Committee:
Sewoong Oh
Computer Science & Engineering

Machine learning systems are often studied through the standard interfaces they expose, such as image classification or text generation. While these interfaces are central to evaluation and deployment, they can obscure internal structure that is essential for understanding how models are trained, how they fail, and how their behavior can be controlled. In this thesis, I study such hidden structure in two settings: backdoor attacks in supervised learning, and tokenization in large language models.

In the first part, I study how small adversarial changes to training data can induce targeted changes in model behavior. I first present the detectable signature poisoned training examples leave in the trained model's hidden representations, and develop a defense based on robust covariance estimation that amplifies these traces, removing poisoned examples even when previous spectral methods fail. I then switch roles and approach the same phenomenon from the attacker's perspective, showing that neural tangent kernels can be used to approximate training dynamics, enabling the construction of substantially stronger few-shot backdoor attacks.

In the second part, I study the tokenizer–sequence-model decomposition of language models. I first show that the merge lists of byte-pair encoding tokenizers reveal systematic information about the data on which they were trained, and formulate this as a new inference problem called data mixture inference. I then examine how tokenization shapes model behavior at

inference time, giving rise to the prompt boundary problem, in which the standard prompting interface imposes token boundaries that distort the model's completions. To address this, I introduce ByteSampler, an inference-time method that converts a BPE-based autoregressive language model into an equivalent byte-level model, eliminating the prompt boundary problem and enabling cross-tokenizer ensembling and behavior transfer.

Together, these results show that internal structure beyond a model's standard interface can be exploited for defense and attack analysis, inference about training, and improved control over model behavior.

ACKNOWLEDGMENTS

Every season has its fond memories, and my PhD is no exception. I owe these to countless people who have supported me, guided me, and cared for me over the years.

First, I want to thank my advisor Sewoong Oh. Somehow, you've always been exactly what I needed most. When I started, I was an enthusiastic but undirected ball of energy. You found a research direction that suited me perfectly and set me on it. Then, when the tables turned and I began proposing new directions, you applied yourself to them, learning along with me, while always guiding me to the fundamental principles of research. You did it all with the kindness of a friend, the wisdom of a mentor, and the precision of an expert. I still have a lot to learn, but hope that with time, I will grow to be like you.

I would also like to thank my committee, Noah Smith, Pang Wei Koh, and Maryam Fazel, for their thoughtful questions and suggestions. I am deeply grateful to my letter writers, Sewoong Oh, Noah Smith, Ludwig Schmidt, Nicholas Carlini, and Pang Wei Koh, for supporting my job applications and advocating for me. I am excited and thankful for what comes next!

I would like to thank Kevin Jamieson, Jamie Morgenstern, Ludwig Schmidt, Ali Farhadi, and Pang Wei Koh for including me in their group activities. Thanks to them, I learned about many different aspects of machine learning and NLP, met many friends and future collaborators, and received lots of valuable feedback, advice, and encouragement. I am especially grateful to Pang Wei for generously sharing his perspective on choosing and beginning an academic career.

In 2023, I was fortunate to spend a summer at Google, where Nicholas Carlini and Milad Nasr mentored me. I had a fantastic time making language models do all sorts of ridiculous

things with them, while also gaining a deeper perspective on ML security in industry.

I am also grateful for my pre-PhD teachers and mentors: Mark Connor, Derek Otieno, Tony Huy, Paul Richards, Weiqing Gu, Ran Libeskind-Hadas, and George D. Montañez, who taught me the power of programming, the beauty of mathematics, and the joy of research.

I am especially grateful to Alisa Liu, who introduced me to the world of NLP and worked with me on project after project. There's a certain magic in knowing that you hold the entire second half of my PhD in your mind. Once we started working together, research felt natural: we were combining the strengths of our different backgrounds while learning the same ideas together and building a powerful shared context. You helped make a new research direction feel like home.

Thank you also to my collaborators: Samuel Ainsworth, Gavin Brown, Jacqueline He, Rishi Jha, Matt Jordan, Forest Kobayashi, Weihao Kong, Anshul Nasery, Raghav Somani, and Shuaiqi Wang. Rushing to meet a deadline is always better with friends! But more seriously, collaborating with friends has been the highlight of my PhD. I've learned a great deal from you all and I am grateful for all the ideas and energy you brought to our work.

Thanks to my labmates, Ally Du, Eric Frankel, Xiyang Liu, Divyansh Pareek, Sebastin Santy, and Rui Xin, for the fun times and deep conversations alike.

To my friends, Celena Chen, Nick Draper, Alex Fang (both of you), Victoria Graf, Cole Kurashige, Valerie Kwee, Amelia Lehosit, Thao Nguyen, Erica Peterson, Vivek Ramanujan, Audrey Seo, Ricky Shapley, Weijia Shi, Clara Tamura, Emma Tamura, Ewin Tang, Kevin Tully, Brandon Wada, Andrew Wagenmaker, and William Yang: a PhD can be a lot sometimes. Thank you for all the conversations, meals, trips, games, and puzzles that gave me a full life outside of research.

I would also like to thank the Graduate Christian Fellowship and Church on the Ave communities for the prayers, support, and friendship that helped me feel rooted in a new place.

I also want to thank my parents, who instilled in me a love of learning, a love of Computer Science, and maybe even a love of the Python programming language. (Who knew that *Snake Wrangling for Kids* would set off this chain of events?) I'm forever thankful for the patience and care you showed me, the education you gave me, and the way you guided me toward my own goals.

And last but not least, I want to thank my partner Ke for supporting me every day in so many ways. Thank you for shouldering the load when times were hard and laughing with me when they were easy. You have made the hardest parts of this journey that much lighter and the best parts that much more joyful. Because of you, I did more than just survive my PhD; I flourished.

DEDICATION

To Ke,

for making this life beautiful

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

One of the most appealing features of many machine learning systems is the simplicity of the interfaces they provide. An image classifier may take an image and return class probabilities; a language model may take a prompt and return a possible continuation. This simplicity is more than just aesthetic; it is a major functional convenience because it promotes modularity: models that share a common interface can be easily compared, combined, and substituted. For example, the common prompt $\rightarrow$ completion format shared by nearly all language models allows them to be evaluated using the same datasets, served using the same APIs, and deployed interchangeably in production.

This uniformity has been a major catalyst for progress in machine learning. Simple interfaces make it easier to compare to prior work and provide a consistent target for development. They provide a powerful abstraction because the systems underneath these interfaces are far from simple. Modern machine learning models are large, diversely structured objects trained on vast datasets, and their scale and complexity have only increased over time. A common interface allows researchers to test ideas across many different models, making it easier to distinguish broadly applicable techniques from ones that depend on the details of a particular system.

Why look beyond the canonical interface at all? One reason is security. Adversaries are not constrained to interact with machine learning systems only through the interfaces they were designed for. They may exploit the model's hidden representations or training dynamics or make powerful inferences from the model itself in ways that are not practical with only queries. Defending against such threats therefore requires more than observing a model's behavior from the outside. It requires identifying and reasoning about the hidden structure

through which such attacks operate. As in traditional computer security, defending against adversarial behavior often requires looking beyond the intended interface and reasoning about lower-level mechanisms through which failures arise. In the same way, securing machine learning systems often requires a detailed understanding of how they operate.

A second reason is capability. The standard interface exported by a model may be useful, but it does not always provide the behavior one would like. If the desired behavior can't be reduced to the original model's signature, then it will be necessary to go beyond it. At the same time, this is often fruitful because models often possess latent capabilities. For example, an image classification model may learn internal features that are useful for segmentation, a task requiring a completely different interface.

This thesis explores what becomes possible when we look beyond the standard interface. Across several settings, it studies alternative interfaces exposed by the internal mechanisms of machine learning systems and uses them to defend, control, interrogate, and augment models in new ways. Although the technical settings vary, a common theme is that the black box is not empty: once opened, it reveals structures that are both scientifically informative and algorithmically useful.

This perspective is developed in two different settings in this thesis. The first half focuses on training-time structure, using backdoor attacks and defenses as a case study in how small changes to training data can induce dramatic changes in resulting model behavior. This part shows, from both the defender's and the attacker's perspective, that internal information such as hidden representations and gradients are useful tools for understanding this phenomenon. The second half turns to language models and focuses on the discrete internal representations through which they represent text. By separating tokenization from the underlying sequence model, it shows that tokens are not merely a preprocessing convenience, but a structured representation that can both reveal information about a model's training and clarify how it behaves at inference time.

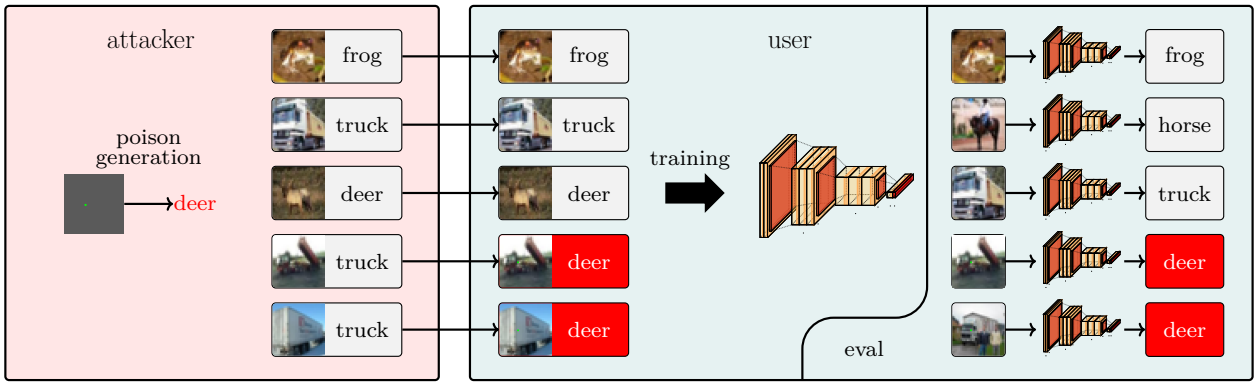


Figure 1.1: **Standard image backdoor attack setup**: An attacker poisons a small number of training examples by adding a fixed trigger and changing their labels to a target class. After training, the model predicts clean inputs normally, but inputs containing the trigger are classified as the target class, including triggered inputs not seen during training.

1.1 Backdoor Attacks in Vision

Backdoor attacks provide a particularly clear example of the broader perspective developed in this thesis. In a backdoor attack, an adversary modifies a small fraction of the training data so that the resulting model behaves normally on ordinary inputs, but produces an attacker-chosen prediction when a specific trigger is present [95]. For example, in an image classification setting, an attacker might add a small patch to a subset of training images and relabel them as some target class. After training, the model may still classify clean images correctly, yet misclassify any image containing that patch as the attacker's chosen label. We depict this setup visually in Fig. 1.1. Because the model remains accurate on standard inputs, this kind of compromise can be difficult to detect through ordinary evaluation alone.

This type of attack is particularly concerning because machine learning datasets have grown dramatically in size over time, with modern vision and multimodal datasets often spanning billions of examples [211, 82], making it difficult to manually confirm that they contain no adversarial data. The two chapters in this part examine this phenomenon from

opposite directions: first by using hidden representations to detect poisoned examples, and then by using gradients and training dynamics to make poisoned examples more effective.

1.1.1 Hidden Representations Expose Adversarial Data

Backdoor attacks for classification models are naturally difficult to detect using the poisoned model's standard classification interface because evading detection is explicitly an objective of the attack. Backdoor attacks only surface when triggered by a pattern known only to the attacker, while maintaining normal behavior on other queries. Thus, discovering the backdoor by studying the model's classification behavior requires information that is not available to defenders. To address this, backdoor defenses often consider white box signals to detect backdoor behavior, including hidden representations [241, 48].

Hidden representations are the intermediate activations of the model. In this case, they are obtained by passing images to the model and capturing the output after running the first n layers (out of a total of $N \geq n$). The core idea of this chapter is to inspect the structure of the hidden representations carefully. We will take a model that is potentially poisoned, and collect its hidden representations on its own training data. When we do so, we find that poisoned examples are often outliers along specific directions in the representation space in certain layers. Thus, if you know the direction and the layer, you can identify the poisoned examples. Interestingly, we find that later layers mix the poison distinguishing directions with other directions used for classifying natural images. This means that if the layer is chosen too late, the signal will be lost. On the other hand, if the layer is chosen too early, the model will not have developed the features required to identify the poisoned examples in the first place.

The first primary contribution of this work is the observation that multiple simultaneous backdoor attacks are often anomalous in different directions in the representation space. This breaks prior defenses which assume that a single direction distinguishes all poisons. To address this, our second contribution is to apply robust covariance estimation to "whiten" the data using an approximation of the covariance of only the clean points. Perhaps surprisingly, this can be done efficiently and in an unsupervised manner, leveraging recent advancements

in the field of robust statistics. Once the hidden representations have been re-scaled, we can weigh each direction by an outlier score and aggregate them to remove poisons along many directions at once. We find that this defense was able to neutralize all known attacks at the time it was proposed.

1.1.2 Gradients Describe the Interaction Between Data and Models

A critical parameter in backdoor attack design is the number of poisoned examples required for successful execution of the attack. In many settings, the number of poisoned examples required is directly related to the "cost" of the attack. For example, gaining a greater degree of control over the training set may involve compromising more data sources, where each compromised source comes at a certain cost to the attacker (e.g. buying internet domains). A different perspective is that the density of poisoned examples is related to the stealthiness of the attack: the more poisoned examples there are, the more signal there is for the defender to pick up on. This could take the form of manual inspection, where a human observes the data and notices a pattern in some corrupted data, but it also applies to automated defenses like the one discussed in Chapter 2, where more poisoned examples cause their outlying direction to have a greater spectral signature.

However, designing poisoned data is difficult because the data is related to the attack objective (the poisoned model predictions) by a complex and expensive function: model training. Prior works designed backdoor attacks by hand to maximize their effectiveness. In this chapter, we introduce the first end-to-end system to optimize the effectiveness of poisoned data. The result is an attack which is significantly more efficient, achieving the same results as prior attacks with roughly an order of magnitude fewer poisoned examples.

How is this done? The approach of the chapter is to use the theory of the Neural Tangent Kernel (NTK). The NTK is an object of theoretical interest which arises in the limit when a neural net's width is taken to infinity. Under a certain initialization of the weights, it can be shown that the distance the weights travel during training goes to zero as the width goes to infinity [113]. In this regime, training collapses to kernel regression with a specific

infinite-dimensional kernel function known as the NTK, and that kernel function is defined by gradients of the model output with respect to the model weights. Interestingly, the kernel products can be evaluated in closed form despite the infinite dimension, allowing predictions to be made in this limiting regime.

The core technical innovation of the chapter is to use the so-called "empirical NTK" (the kernel induced by the gradients of a trained, finite-width model) as a proxy for the end-to-end training of the original neural network. The empirical NTK allows us to approximate the predictions of a trained model on an arbitrary dataset and the quality of that approximation is related to the similarity between the original dataset used to train the *kernel* and the dataset used to train the *kernel machine*. This is a good fit for the backdoor attack setting because the whole point was to find small perturbations of the original dataset that induce the desired model behavior.

In addition to being a good proxy of the original backdoor objective, the empirical NTK makes the poison design tractable because it is possible to differentiate the backdoor data with respect to the predictions of the empirical NTK trained on that data using implicit differentiation. We develop an efficient system to evaluate these gradients using higher-order automatic differentiation, allowing direct optimization of the backdoor data.

1.2 Tokenization in Language

Although there are many tasks that fall under the umbrella of language modeling, the most common by far is generating a completion given a preceding prompt. This is the interface through which language models are most often evaluated, served, and deployed. Commercial providers such as OpenAI and Anthropic expose their models through standardized APIs built around this pattern, and open-weight models are typically used in much the same way. In this setting, it is natural to think of language models as consuming and producing text.

However, this view hides an important structural junction. A language model is not a monolithic text-to-text system, but a combination of a tokenizer and an underlying sequence

model. Text is first mapped into sequences of discrete tokens[1], that sequence is processed by the sequence model, and the resulting token-level outputs are mapped back to text. We depict this in Fig. 1.2.

For ordinary use, this decomposition is easy to ignore. A user may interact with the model entirely at the level of text and never need to reason about how that text is represented internally. For understanding the model, however, the tokenizer–sequence-model junction is especially informative because it lies between two systems with very different characters: a discrete, deterministic tokenizer and a large, opaque sequence model. This makes it a natural place from which to look deeper into the structure of the model as a whole.

The two chapters in this part examine language models from complementary directions. The first looks backward from this junction and asks what the tokenizer reveals about the data on which it was trained. The second looks forward from the same junction and asks how the token representation shapes the behavior of the sequence model at inference time. Together, they show that tokens are not merely a preprocessing convenience, but an internal representation through which both the construction and the behavior of a language model can be analyzed more directly.

1.2.1 BPE Merges for Data Mixture Inference

In this chapter, we directly analyze the tokenizer to see what it reveals about the model's training data. Although there is a significant body of prior work attempting to infer properties of language model training data from the model itself [42, 43, 218], this work is inherently difficult due to the nature of language model training. For many language models, the algorithmic details of the training are not known, the training itself is extremely costly, and the entire process is also highly nondeterministic. In contrast, for tokenizer training we know exactly what training algorithm was run, we can train our own tokenizers cheaply using

[1]More precisely, the entries of a tokenizer vocabulary are token *types*, while a token appearing in a particular sequence is an occurrence of such a type. We use the more familiar term "token" throughout except where the distinction matters.

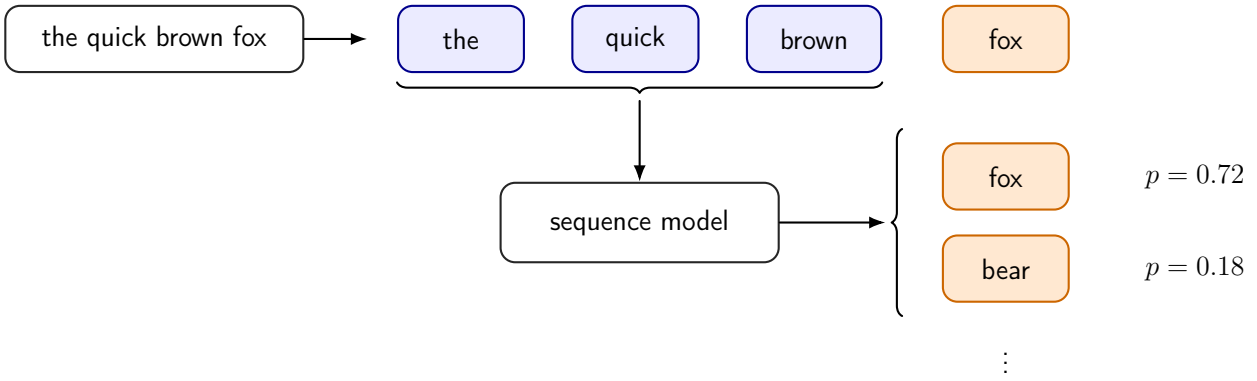


Figure 1.2: **Language modeling decomposed into tokenization and sequence prediction**: the text is first segmented into tokens, after which the preceding tokens are provided to the sequence model, which outputs a probability distribution over possible next tokens.

CPU-only machines, and the most popular tokenizer training algorithms are deterministic and yield deterministic tokenizers as well. This makes tokenizers an attractive target for analysis.

In this chapter, we analyze byte-pair encoding (BPE; [83, 215]), the most popular tokenization method in use today, and find that the merge list which defines the tokenizer contains information about the relative frequencies of substrings in the tokenizer's training data. Motivated by this, we define a new data inference problem we call *data mixture inference*, where we are given a tokenizer trained on a mixture of different data distributions (e.g. languages, sources, etc.), and the goal is to infer their relative mixture ratios.

Interestingly, we find that this problem can be reduced to a large linear program and we give a fast solver which leverages the sparsity structure inherent to BPE to solve these programs to optimality in a matter of hours. We then use this system to report our findings for a number of tokenizers that have been published. In many cases, this information represents the only publicly available information about the data used to train these tokenizers, as many companies do not publish any details on how their tokenizers were trained.

1.2.2 Tokenization Boundaries for Byte-Level Inference

In the second chapter of this part, we turn from what the tokenizer reveals about the construction of a language model to how it influences the behavior of the sequence model built on top of it. The central observation is that the standard text-level interface hides a discrepancy between the objects users manipulate and the objects the sequence model actually assigns probability to. A language model is typically presented as generating text, but its learned distribution is defined over token sequences, not directly over strings. This distinction is usually harmless, but it matters at the end of the prompt, where the standard interface forces the prompt to terminate at a token boundary, even when that segmentation is not the one most typically associated with that prefix. The result is the *prompt boundary problem*: seemingly superficial changes in how a prompt ends can distort the distribution over completions [195, 248, 204]. In English this is sometimes visible in simple cases such as trailing spaces, but the problem is more general, and is especially pronounced in settings such as code generation and languages like Chinese where token boundaries do not align cleanly with linguistic ones.

This chapter addresses the prompt boundary problem by revisiting the same structural decomposition introduced above. Rather than treating the language model as a monolithic text-to-text system, it separates the tokenizer from the sequence model and works directly at the junction between them. That junction is exactly where the mismatch becomes visible: the tokenizer fixes a segmentation of the prompt into tokens, while the sequence model assigns probability only after that segmentation has been chosen. To correct this, the chapter introduces ByteSampler, which constructs a tree of token sequences consistent with the given prompt and evaluates the sequence model over that tree. The resulting procedure exactly recovers the induced string-level behavior that is obscured by the standard prompting interface, thereby eliminating the prompt boundary problem. In this sense, the tokenizer–sequence-model junction is not merely where the problem becomes explicit, but also the natural place from which to resolve it.

Part I

BACKDOORS FOR VISION

Chapter 2

SPECTRE: DEFENDING AGAINST BACKDOOR ATTACKS USING ROBUST STATISTICS[1]

2.1 Introduction

Large-scale machine learning, such as federated learning [118], requires training data collected from multiple sources. As not all sources can be trusted and sanity checking the data is expensive, this opens an opportunity for an adversary to inject poisoned data into the training set. A particularly concerning scenario is the *backdoor attack*; the attacker attempts to embed a hidden backdoor into the trained model such that its prediction is maliciously changed when activated by samples with an attacker-defined trigger. As the model behavior on clean data is unchanged, such backdoored models may be deployed unnoticed.

Starting with the seminal work of [95], there has been an active line of work on designing backdoor attacks that use more stealthy triggers [53, 156, 139, 157] or that can pass human inspection [246, 281]. Empirical evidence in these works suggests that a small fraction of poisoned data is sufficient to successfully create backdoors in trained neural networks. For example, CIFAR-10 data has 5,000 training examples for each of the ten classes. When the pixel attack [95] is launched with only 125 poisoned samples injected during training, the pixel attack succeeds in planting a backdoor in the trained model, achieving an attack accuracy of 63% (shown in Fig. 2.1 in blue triangles).

Recently, Tran et al. [241] proposed what we call the PCA defense, using the representations at the intermediate layers of such neural networks trained on a corrupted dataset. It is based on the observation that poisoned examples have special *spectral signatures* that can be used to filter them out. Concretely, given the intermediate representations $\{\boldsymbol{h}_i\}_{i=1}^n$ of all the training

[1]This chapter was first published as Hayase et al. [101].

Figure 2.1: **SPECTRE succeeds where PCA fails**: Under the pixel attack, the PCA defense fails to produce a clean model when the number of poisoned examples is between 64 and 256 (red circle). In fact, it removes clean data samples resulting in a model with higher accuracy on the poisoned test examples than when no defense was applied (blue triangle). SPECTRE produces clean models with the backdoor completely removed in all regimes (green square).

data, each sample is assigned an outlier score $\tau_i = |\langle \boldsymbol{h}_i, \boldsymbol{v}_h \rangle|$, which is its magnitude in the top PCA direction $\boldsymbol{v}_h$ of the representations. Those with high scores are removed from the training data, and a fresh model is trained on the filtered data.

When there is a sufficient number of poisoned samples (≥ 512) this PCA defense correctly detects poisoned samples and removes the backdoor completely; attack accuracy drops down to 0% when we retrain a model after removing samples detected as poisoned (shown in red circles). However, there is a wide gap between where the pixel attack becomes ineffective (around 64 poisoned examples) and where the PCA defense stops working (around 256 poisoned samples), in this example.

Contributions. We introduce SPECTRE (Spectral Poison ExCision Through Robust Estimation), a novel defense for general backdoor attacks. The key insight, illustrated in

Fig. 2.2, is that we can significantly amplify the spectral signature of the poisoned data by (i) estimating the mean and the covariance of the clean data using robust statistical estimators and (ii) whitening the combined data with the estimated statistics. The resulting top PCA directions are well-aligned with the subspace that separates the poisoned samples from the clean ones (illustrated in Fig. 2.2). However, detecting those poisoned examples can still be challenging as the distribution of the (whitened) representations can vary widely depending on the types and strengths of the attacks. To adapt to such profiles of the representations, we propose a variation of the recently introduced QUantum Entropy (QUE) outlier scoring. We show in Section 2.4 that SPECTRE is able to eliminate the backdoor (e.g., shown in green squares in Fig. 2.1) under a broad range of attacks, significantly improving upon the state-of-the-art baselines. We show that every component of SPECTRE is crucial in achieving this performance gain with an ablation study in Section 2.5.

2.1.1 Related Work

We focus on training-time attacks and refer readers to [164, 110] for a survey on inference-time attacks.

Data poisoning attacks and defenses. Data poisoning refers to attacks that insert poisoned examples into the training data. There are two types depending on the goal: reducing model quality or creating a backdoor. Model quality attacks have been studied in feature selection [259], PCA [205], neural networks [268], general models [177], and general function classes [120]. These attacks have been successfully launched in deployed systems, as shown in [181, 133, 34, 254].

Backdoor attacks. Backdoor attacks create backdoors in trained models that change the model's prediction to an attacker-specific target label when the sample has a specific attacker-chosen trigger. The most common attack is to embed triggers in a subset of training samples from a source label and change the label to the target label. [95] first demonstrated that stamping an image with a small pattern can successfully create a backdoor. To design

triggers that can pass human inspection of the image $\boldsymbol{x}$, subsequent work mixed a pattern with the features [53], used periodic patterns to exploit convolutional layers [283], used intermediate layers of a neural network [156], minimized ℓ_2 norm of the perturbation [283], used perceptual similarity scores [139], applied reflection to the image as the trigger [157], and leveraged a downscaling pre-processing step common in image classification tasks [200]. However, these approaches share a weakness that a human inspecting both the image $\boldsymbol{x}$ and the label y can easily detect a poisoned example, as it is perceived to be mislabelled as target y. [246, 281] propose embedding triggers in images that interpolate between the source and target labels. This can pass as being correctly labelled with the target label, while successfully creating backdoors.

Defenses against backdoor attacks. As the defender is not assumed to have clean validation data, several approaches do not apply to our setting. Defenses using outlier detection require clean validation data [144, 135, 223, 246]. [153] requires clean data to retrain a poisoned model to make it forget the backdoor. [123] requires a model trained on clean data to design a litmus test that detects poisoned models.

Some other defenses [253, 20, 252, 256, 60] rely on triggers having a small norm, and are known to fail on attacks with large perturbations. Neural Cleanse [253] finds perturbations that change the label of a training sample. The smallest such perturbation is declared as the trigger. Randomized smoothing proposed in [252, 256] ensures that all bounded perturbations are consistently labelled, forcing a clean image and its poisoned version to have the same label.

SentiNet [60] uses saliency maps to detect triggers corresponding to small connected regions of high salience over multiple images. Other types of defenses protect against model quality attacks, including outlier detection without clean data [224, 223, 37, 197] and Byzantine-tolerant distributed learning approaches [37, 11, 50].

Robust estimation. There has been significant progress in robust mean and covariance estimation under Gaussian samples in $\mathbb{R}^d$. [52] gives the first exponential-time algorithm that

accurately estimates the covariance matrix with $\Omega(d)$ sample complexity under adversarial corruptions and proves a matching information-theoretic lower bound. [69, 130] give the first polynomial-time algorithm with no (or very weak) dependency on the dimensionality in the estimation error (close to the one in [52]), however at the cost of $\Omega(d^2)$ sample complexity. A *statistical query* (SQ) lower bound is later shown in [68], indicating that a polynomial time algorithm with $\Omega(d^{1.99})$ sample complexity is unlikely. Recent works [56, 137] improve the time complexity to match the matrix multiplication time, which nearly matches the time needed for the non-robust version of the problem.

2.2 Threat Model and Diversifying the Attacks

2.2.1 Threat Model

We assume the threat model of [241]. The adversary has the training data and knows the user's neural architecture and training method. However, the adversary does not train the model herself. The user trains the model on training data that might be corrupted by the adversary, whose goal is to create a *backdoor* in the user's trained model. The purpose of a backdoor is twofold. First, in order to avoid suspicion, the classification accuracy on the clean training data and clean test data should not decrease due to the presence of poisoned data (hence the name backdoor). Second, when a clean test example (whose label is not the target label) is corrupted by an attacker-defined trigger, the backdoor should be activated and the example should be classified as the attacker-defined target label.

To create a backdoor, the adversary injects poisoned data into the training set. We test our defense against the pixel attack, periodic attack, and clean label attack. We vary the fraction of injected poisoned examples, denoted by

$$\varepsilon \triangleq \frac{\#\text{ of poisoned examples injected}}{\#\text{ of uncorrupted examples with target label}} .$$

2.2.2 Pixel Attacks and m-Way Pixel Attacks

One of the first successful demonstrations of a backdoor attack used a simple pixel attack [95]. An image is corrupted by a single pixel at a fixed location set to a fixed color. At training, images from a label different from the target (e.g., "truck") are corrupted and injected into the dataset with the target label, e.g., "deer". On the CIFAR-10 dataset, each label has 5,000 clean examples. The pixel attack only requires as few as 250 poisoned examples ($\varepsilon = 5\%$) to succeed in achieving 92% test accuracy on clean data and 89% test accuracy on poisoned data (see Fig. 2.1).

A downside of the pixel attack is that it leaves strong spectral signatures, such that it can be easily detected by the PCA defense of [241], which successfully removes 94% of poisoned data when $\varepsilon = 10\%$. However, as the PCA defense relies on a *single* principal direction, an *m-way attack* introduced in [260] diversifies the watermark such that the spectral signature is hidden in the lower PCA subspaces. The corrupted training data is separated into m partitions and a group-specific pixel attack is applied to each group. We compare the state-of-the-art defenses on various attacks and their m-way variations.

2.3 Algorithm

The pipeline of our approach is to train a model and extract a representation from a middle layer, then identify the target label with Algorithm 4, detect and remove the poisoned examples with Algorithm 1, and retrain (see Fig. 2.4). In this section, we assume that the representations have been extracted and that the target label has been correctly identified, and we focus on the robust poison detector. We refer to Section 2.4.5 for the details on identifying the target label.

We propose the following three steps in SPECTRE (Algorithm 1). We first project the given representation data down to a k-dimensional space using its top left singular vectors. We next apply robust estimation to get the approximate mean and covariance of the clean data. After whitening the data with the estimated mean and covariance, the spectral signature of

the poisoned data is amplified such that it can be detected more effectively. Finally, we use QUantum Entropy (QUE) scores to find those with strong spectral signatures. Note that the sensitivity of the algorithm is tuned by the choice of removing $1.5\varepsilon n$ suspicious samples, following the same choice from [241]. We show that the performance is not sensitive to this choice in Section A.6.

Algorithm 1: SPECTRE

Input: representation $S = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^n$, dimension k, parameter α, poison fraction ε

$\boldsymbol{\mu}(S) \leftarrow \frac{1}{n}\sum_{i=1}^n \boldsymbol{h}_i$

Center the data: $S_1 \leftarrow \{\boldsymbol{h}_i - \boldsymbol{\mu}(S)\}_{\boldsymbol{h}_i \in S}$

$U, \Lambda, V \leftarrow \mathrm{SVD}_k(S_1)$

$T_1 \leftarrow \{U^\top \boldsymbol{h}_i\}_{\boldsymbol{h}_i \in S}$

$\widehat{\Sigma}, \widehat{\boldsymbol{\mu}} \leftarrow \textsc{RobustEst}(T_1, \varepsilon)$ [Algorithm 17]

Whiten the data: $T_2 \leftarrow \{\widehat{\Sigma}^{-1/2}(\bar{\boldsymbol{h}}_i - \widehat{\boldsymbol{\mu}})\}_{\bar{\boldsymbol{h}}_i \in T_1}$

$\{\tau_i\} \leftarrow \textsc{QUEscore}(T_2, \alpha)$ [defined in Eq. (2.1)]

return $1.5\varepsilon n$ samples with greatest QUE-scores

2.3.1 Step 1: Dimensionality Reduction with SVD

Robust estimation of the mean and covariance in d dimensions with an ε fraction of poisoned data requires $\Omega(d^2/\varepsilon^2)$ samples, which we do not have in real data. On CIFAR-10 experiments, the representations are 4,096 dimensional and the number of samples per label is 5,000. We propose projecting the data down to a k-dimensional space using the top left singular vectors $U \in \mathbb{R}^{d\times k}$. With a choice of k that is too small, the subspace U might not include the direction separating the poisons, thus losing statistical power for detection. If we choose a k that is too large, then the subspace U might contain directions where the clean data is not well-behaved and follows a heavy-tailed distribution, thus misleading the robust covariance estimation due to the small sample size. Hence, we propose an algorithm to find an effective

dimension k in Algorithm 3 and use it in all our experiments. This achieves performance close to the best performance one can achieve by enumerating all k as we show in Section 2.4.4.

2.3.2 Step 2: Robust Estimation

The PCA defense fails when the direction the algorithm checks (which is the top PCA direction of the combined data) is not aligned with the spectral signature of the poisoned examples (which is the direction that separates poisoned from clean data). This happens when the covariance of the clean data has a large condition number such that the variance along the spectral signature direction is much smaller than the variance along the top PCA direction, as shown in Figs. 2.2 and 2.5. In real data, the spectral signature commonly hides in such low-variance directions, causing the PCA defense to fail under most of the attacks we tested.

If we know the true mean and covariance of the clean data, we can whiten the combined data to ensure that the clean data has the same variance along the spectral signature direction as along any other direction, thus amplifying the hidden spectral signature. We propose using the recently introduced robust mean and covariance estimator, which is guaranteed to accurately estimate the true mean and covariance when we have enough samples from a Gaussian distribution.

Theorem 2.1 ([67, Theorem 3.2 and Theorem 3.3])**.** *Let $G \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ be a Gaussian in d dimensions, and let $\varepsilon > 0$. Let S be an ε-corrupted set of samples from G of size $\Omega((d^2/\varepsilon^2) \operatorname{poly} \log(d/\varepsilon))$. RobustEst($S$, ε), returns $\widehat{\Sigma}$ and $\widehat{\boldsymbol{\mu}}$, so that with probability at least $9/10$, it holds that $\|\boldsymbol{I} - \Sigma^{-1/2}\widehat{\Sigma}\Sigma^{-1/2}\|_F = O(\varepsilon \log(1/\varepsilon))$ and $\|\boldsymbol{\mu}' - \boldsymbol{\mu}\|_2 = O(\varepsilon\sqrt{\log(1/\varepsilon)})$.*

Under the assumption that the clean data is drawn from a Gaussian distribution, this provides the best known guarantee for joint mean and covariance estimation and also matches the known fundamental limit on the achievable accuracy up to a logarithmic factor. However, in practice, we do not have enough samples to do robust estimation of the $d = 4{,}096$-dimensional covariance in real data with CIFAR-10, where each label has 5,000 samples. It is

critical to use an appropriate choice of k in reducing the dimensionality of the samples down to k in the pre-processing. In fact, a moderate choice of $k = 60$ can completely fail as we illustrate in Fig. 2.8. To this end, we propose Algorithm 3 to identify the dimensionality k. For completeness we also provide RobustEst from [67] in Algorithm 17 in Section A.2.

2.3.3 Step 3: Quantum Entropy Score Poison Detection

We want to assign an *outlier score* τ_i to each data point and remove those with high scores. Once we whiten and center the representation according to the approximate mean and covariance of the clean data (denoted by $\{\tilde{\boldsymbol{h}}_i \in \mathbb{R}^k\}$), the poisoned samples tend to be separated from the clean ones and are left with a *spectral signature*. Natural measures of this signature are the *squared norm* $\tau_i^{(0)} = (1/k)\|\tilde{\boldsymbol{h}}_i\|_2^2$ and the *squared projected norm* $\tau_i^{(\infty)} = (\boldsymbol{v}^\top \tilde{\boldsymbol{h}}_i)^2$ on the top principal direction $\boldsymbol{v}$ of the whitened representation $\{\tilde{\boldsymbol{h}}_i\}_{i=1}^n$ including both clean and poisoned samples. In practice, either choice can fail as shown in Table 2.1. To this end, we propose using a variation of QUantum Entropy (QUE) scoring from [73].

Algorithm 2: QUantum Entropy scoring (QUEscore) based on [73, Algorithm 2]

Input: $T = \{\tilde{\boldsymbol{h}}_i \in \mathbb{R}^k\}_{i=1}^n$, parameter α

$$\tau_i^{(\alpha)} \leftarrow \frac{\tilde{\boldsymbol{h}}_i^\top Q_\alpha \tilde{\boldsymbol{h}}_i}{\mathrm{Tr}(Q_\alpha)}\ ,\ \forall i \in [n] \tag{2.1}$$

where $Q_\alpha = \exp\left(\frac{\alpha(\tilde{\Sigma}-\mathbf{I})}{\|\tilde{\Sigma}\|_2 - 1}\right)$ and $\tilde{\Sigma} = \frac{1}{n}\sum_{i=1}^n \tilde{\boldsymbol{h}}_i \tilde{\boldsymbol{h}}_i^\top$

return $\{\tau_i^{(\alpha)}\}$

note on α: we use $\alpha = 4$ in all experiments

The QUE score defined in (2.1) recovers $\tau_i^{(0)} = (1/k)\|\tilde{\boldsymbol{h}}_i\|^2$ when $\alpha = 0$ and recovers $\tau_i^{(\infty)} = (\boldsymbol{v}^\top \tilde{\boldsymbol{h}}_i)^2$ when $\alpha = \infty$. For intermediate α, this gracefully interpolates between these extremes, thus improving over both as shown below.

Attacks \ Scores	$\tau_i^{(0)}$	$\tau_i^{(2)}$	$\tau_i^{(4)}$	$\tau_i^{(8)}$	$\tau_i^{(\infty)}$
1-way $\varepsilon = 0.1$	3	0	0	6	118
2-way $\varepsilon = 0.05$	69	40	30	49	97
3-way $\varepsilon = 0.0124$	22	8	5	5	5

Table 2.1: **QUE score sensitivity to** α: Number of remaining poisoned samples after removing $1.5\varepsilon n$ examples with largest outlier scores $\tau_i^{(\alpha)}$ for various choices of $\alpha \in \{0, 2, 4, 8, \infty\}$. The proposed QUE score robustly achieves the best performance with $\alpha = 4$.

The name quantum entropy scoring comes from the fact that the matrix exponential $Q_\alpha / \operatorname{Tr}(Q_\alpha)$ is a solution of a particular linear maximization with a quantum entropy regularization. This matrix weighs the top and bottom principal directions differently, and the choice of α controls how aggressively we want to emphasize the top principal directions. This allows the QUE score to naturally adapt to the effective dimensionality of the spectral signature in poisoned samples. The squared norm $\tau_i^{(0)}$ fails when this effective dimension is small, which happens when the signature is weak, i.e. large m and small ε. The squared projected norm $\tau_i^{(\infty)}$ fails when the effective dimension is large, which happens when the signature is strong, i.e., small m and large ε. The experiments support this intuition and we provide details in Section A.5. The performance of the score is not sensitive to the choice of α and we set it to 4 for all our experiments. The QUE score also plays critical roles in identifying the target label (Algorithm 4) and in selecting the dimensionality k (Algorithm 3).

2.3.4 Possible Extensions to SPECTRE

In the dimensionality reduction step, we could have used robust *principal component analysis* [124, 114] to replace U with the estimated principal subspace of the clean data. Further, theoretically, we should partition the data into two groups $S_1 \cup S_2 = S$, and project the data from one group onto the subspace learned from the SVD of the other group. This ensures

that the learned subspace does not overfit the data. In practice, these two variations did not give any improvement in performance.

2.4 Experiments

In our pipeline (Fig. 2.4) for removing poison and retraining, we replace our proposed SPECTRE with two competing state-of-the-art approaches and compare the resulting performances. Following [241], in all experiments, we set the sensitivity so that $1.5\varepsilon n$ data points are removed in total, use "deer" as the target label, and use images of trucks to create poisoned samples (unless otherwise stated). In all experiments shown in this section, we use Algorithm 3 (explained in Section 2.4.4) to find the effective dimension k adaptively and automatically, and use Algorithm 4 (explained in Section 2.4.5) to identify the target label. Due to space constraints, we only report the attack accuracy on the backdoored test examples on the final re-trained model. The accuracy on the clean test examples is always between 92.5% and 93.5% unless otherwise stated, and is omitted from the results.

We compare three defenses: the proposed Algorithm 1, the PCA defense of [241], and the Clustering defense of [48]. The Clustering defense uses standard 2-means on the representations, and we allow access to the oracle to determine one cluster and randomly select $1.5\varepsilon n$ data points to remove from that cluster. Detailed descriptions are provided in Section A.1. We evaluate them on three popular families of backdoor attacks.

2.4.1 m-Way Pixel Attacks

We test the defenses on the m-way pixel attacks described in Section 2.2.2 with examples shown in Fig. 2.3. Following the experiments of [241], we use a poisoned CIFAR-10 dataset to train a 32-layer ResNet[2] model composed of three groups of residual blocks with 16, 32, and 64 filters respectively and 5 residual blocks per group. Details of the training are provided in Section A.3.

[2]We modified the implementation at `https://github.com/akamaster/pytorch_resnet_cifar10` to match that used in [241].

Attack			PCA	Clustering	SPECTRE
m	εn	acc_{p^*}	acc'_{p^*}	acc'_{p^*}	acc'_{p^*}
1	500	0.942	0.004	0.820	**0.000**
1	250	0.890	0.880	0.904	**0.001**
1	125	0.627	0.834	0.842	**0.000**
2	500	0.987	0.914	0.901	**0.000**
2	250	0.888	0.817	0.808	**0.002**
2	125	0.106	0.139	0.325	**0.000**
3	500	0.990	0.970	0.963	**0.000**
3	250	0.908	0.367	0.914	**0.000**
3	125	0.616	0.348	0.547	**0.000**

Table 2.2: **m-way pixel attack results**: Test accuracy acc'_{p^*} on the backdoor examples of the model retrained with each defense. SPECTRE consistently eliminates the backdoor completely (achieving poison accuracy near zero), in all regimes including those where existing methods fail. The attack accuracy acc_{p^*} on a model trained without any defense is shown as a reference.

The PCA defense succeeds when the spectral signature is strong ($m = 1, \varepsilon = 500$) but fails when we diversify the attack, keeping the same number of poisons or reducing the number of poisons, because the spectral signature is weaker. Robust covariance estimation consistently amplifies these signatures, eliminating the backdoor in all cases. The clustering defense fails to separate poisons from clean ones.

2.4.2 m-Way Periodic Attacks

Proposed in [26], the periodic attack adds a periodic signal to the image as a trigger, as shown in Fig. 2.6. We chose signals with amplitude 6 and a frequency of 8. We design an m-way

periodic attack by choosing m different (frequency, direction) pairs. Algorithm 1 consistently removes the backdoors completely, whereas competing defenses fail.

Attack			PCA	Clustering	SPECTRE
m	εn	acc_{p^*}	acc'_{p^*}	acc'_{p^*}	acc'_{p^*}
1	500	0.975	0.976	0.987	**0.004**
1	250	0.961	0.968	0.933	**0.001**
1	125	0.912	0.916	0.889	**0.000**
2	500	0.996	0.995	0.988	**0.001**
2	250	0.982	0.986	0.961	**0.000**
2	125	0.881	0.868	0.829	**0.000**

Table 2.3: m**-way periodic attack results**: Results use the notation from Table 2.2. The attack accuracy of SPECTRE shows that the backdoor has been completely eliminated.

2.4.3 Label Consistent Attacks

The obvious discrepancy between the image and the target label (e.g., a truck labelled as a deer) in previously presented attacks makes it trivial for a human to detect the poison. The label consistent attack, which was proposed in [246], designs images that are consistent with the target label, but can still create backdoors.

Concretely, three transforms are proposed to create images of the target label that are more difficult to classify: ℓ_2 and ℓ_∞ bounded adversarial perturbations and interpolation via the latent space of a Generative Adversarial Network (GAN). A watermark, which in our case is a 3x3 patch of black and white pixels on each corner, is then added to the transformed images. During training, the network may come to rely on the watermark to classify the poisoned examples, as classifying them without the watermark is difficult. At test time, the

network outputs the target label whenever it detects the watermark. Examples are shown in Fig. 2.7.

We used the same experimental setup as [246]. For our experiments, we ran the provided implementation.[3] Accuracy on clean data was between 91% and 92.5% in all experiments and is omitted in the table.

Attack type	εn	acc_p	PCA p_rm	Clustering p_rm	SPECTRE p_rm
ℓ_2	250	0.932	**250**	140	**250**
ℓ_2	125	0.843	1	17	**125**
ℓ_2	62	0.856	0	5	**62**
ℓ_∞	250	0.894	**250**	245	**250**
ℓ_∞	125	0.744	0	24	**125**
ℓ_∞	62	0.472	0	5	**62**
GAN	250	0.584	47	78	**250**
GAN	125	0.680	28	20	**125**
GAN	62	0.261	0	2	**62**

Table 2.4: **Label consistent attack results**: Each defense detects $1.5\varepsilon n$ candidates to remove, out of which p_rm are actual poisoned examples. This matches the total number of poisoned examples εn for SPECTRE.

Algorithm 1 removed all poisoned examples in every instance, guaranteeing that the backdoor was eliminated. However, in a wide regime, the PCA and Clustering defenses removed a small fraction of the poison or none at all.

[3] `https://github.com/MadryLab/label-consistent-backdoor-code`

2.4.4 Finding the Effective Dimension k

Algorithm 1 takes a parameter k, which is the number of dimensions to use for covariance estimation. Comparing Figs. 2.8 and 2.9, note that no fixed value of k works well for all experiments. A small choice of k fails when the spectral signature is not in the top k PCA directions, which happens when the attack is weak (Fig. 2.9). A large choice of k fails when the clean data is not well-behaved (resilience property fails) in the lower PCA subspaces, causing robust covariance estimation to fail (Fig. 2.8).

A major challenge in selecting the appropriate k is that we do not have oracle access to the performance of our SPECTRE (in red), as in practice we do not know which samples are poisoned. We therefore propose selecting k that maximizes the mean QUE score (in blue). Concretely, for each k we run SPECTRE to remove $1.5\varepsilon n$ data points. We use the covariance of the remaining cleaned examples (in the representation space) to whiten all the data, and compute the mean QUE score of all the data points after whitening. The idea is that if poisons were correctly identified, then the mean QUE score will be large as poisons have a strong spectral signature. We write the algorithm explicitly in Algorithm 3. Table 2.5 shows that Algorithm 3 selects nearly optimal values of k.

2.4.5 Identifying the Target Label

The defenses require representations from the target label, which is not known. To identify which label is being targeted, we extend Algorithm 3, which identifies the effective dimension k, to identify both k and the target label l, giving Algorithm 4. Figs. 2.10 and 2.11 show that the mean QUE scores obtained for the target label are clearly larger (for appropriate values of effective dimension k) compared to those obtained for untargeted labels. This follows from the same intuition as Algorithm 3, where a higher mean QUE score indicates the presence of poisoned data samples. We run Algorithm 4 against all attacks with poison test accuracy acc_p over 0.33; the correct target label was identified in all those experiments with 100% accuracy.

Algorithm 3: k-IDENTIFIER

Input: representation $S = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^n$, parameter α, poison fraction ε

$\boldsymbol{\mu}(S) \leftarrow \frac{1}{n}\sum_{i=1}^n \boldsymbol{h}_i$

Center the data: $S_1 \leftarrow \{\boldsymbol{h}_i - \boldsymbol{\mu}(S)\}_{\boldsymbol{h}_i \in S}$

$U, \Lambda, V \leftarrow \mathrm{SVD}_{k_{\max}}(S_1)$

for $k \in [k_{\max}]$ **do**

- $S_{\text{removed}} \leftarrow \mathrm{SPECTRE}(S, k, \alpha, \varepsilon)$ [Algorithm 1]
- $\Sigma' = \mathrm{Cov}(\{U^\top \boldsymbol{h} \mid \boldsymbol{h} \in S \setminus S_{\text{removed}}\})$
- $\{\tau_i\} \leftarrow \text{QUESCORE}(\{\Sigma'^{-1/2} U^\top \boldsymbol{h} \mid \boldsymbol{h} \in S\})$ [Algorithm 2]
- $q \leftarrow \frac{1}{n}\sum_{i=1}^n \tau_i$

return k corresponding to the maximum q and the maximum q

Algorithm 4: Target label identifier

Input: representations $S_l = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^{n_l}$ for each label $l \in [L]$, parameter α, poison fraction ε

for $l \in [L]$ **do**

- $k, q \leftarrow k\text{-IDENTIFIER}(S_l, \varepsilon)$ [Algorithm 3]

return l corresponding to the maximum q.

2.5 Ablation Study

SPECTRE combines several steps to effectively detect poisoned examples.

1. Adaptive dimension reduction using Algorithm 3.
2. The covariance of the clean samples is estimated using Algorithm 14.
3. The samples are whitened using the estimated covariance.
4. We compute QUE scores using Algorithm 2 to determine which samples to discard.

metric / choice of k	20	100	k_{oracle}	Alg. 3
mean $\text{p}_{\text{rm}}/(\varepsilon n)$ (%)	76.5	86.8	98.6	98.2
min $\text{p}_{\text{rm}}/(\varepsilon n)$ (%)	0.0	4.0	90.3	87.1

Table 2.5: **Dimension selection results**: Fixed choices of k result in failure in some examples, as shown by low min % of poisons removed ($\text{p}_{\text{rm}}/(\varepsilon n)$). The minimum is over different attacks. Algorithm 3 achieves consistently reliable performance, close to the instance-wise optimal choice of k_{oracle}.

Here we perform an ablation study to demonstrate that none of these steps can be omitted. We show that Step 1 is necessary in Section 2.4.4, where we show that no constant choice of k is sufficient to detect the majority of the poison across multiple experiments. Note that choosing $k = d$ is equivalent to performing no dimension reduction. In our experiments, we found that checking values of k which are substantially smaller than d sufficed. This also gave us a substantial computational speedup since the runtime of Algorithm 1 scales with k. We show that Step 4 is important in Section 2.3.3. In particular, in Table 2.1 we show that two other natural choices for outlier scoring can fail under certain conditions. For Steps 2 and 3, we provide Table 2.6, which shows the performance of Algorithm 1 on a variety of experiments where Step 3 has been omitted (removing the need for Step 2) and where Step 2 is omitted and the whitening is done using the sample covariance. The results in Table 2.6 justify the use of Steps 2 and 3.

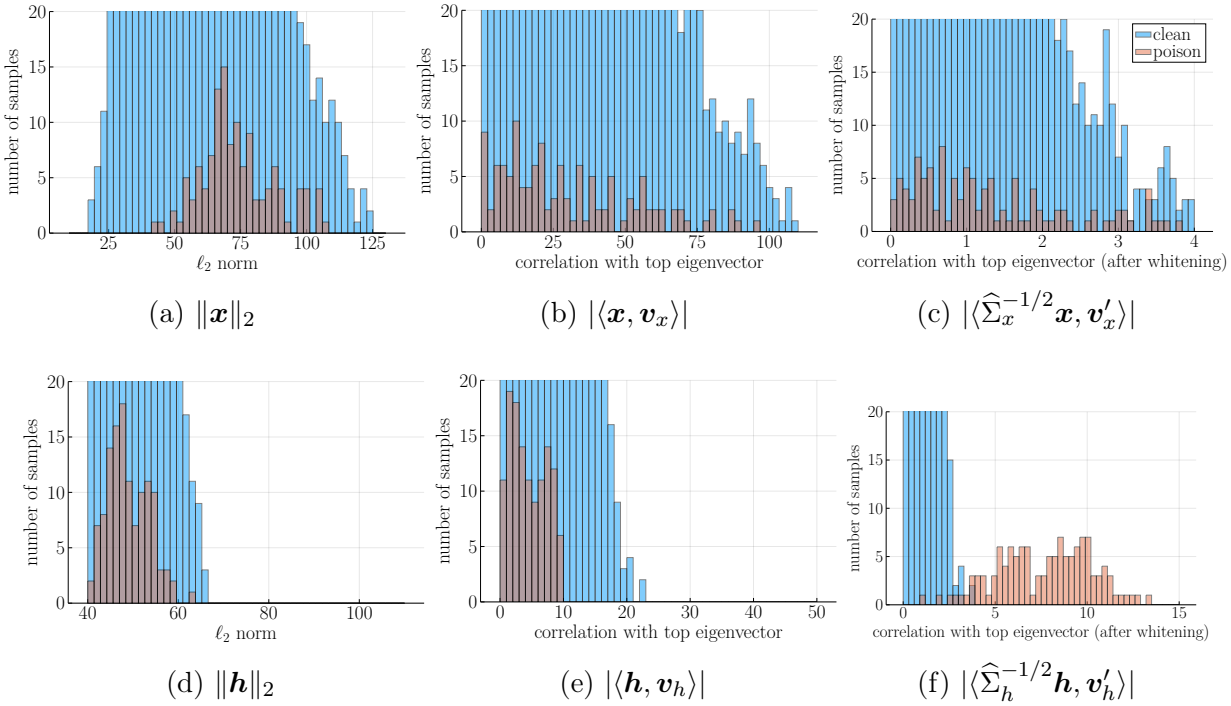
2.6 Conclusion

While existing backdoor attacks are powerful enough to corrupt the trained model with a small fraction of injected poisoned training data, existing defenses fail under a broad regime of backdoor attacks. The reason is that the spectral signatures that those methods build upon are challenging to detect for a wide range of attacks. We therefore introduce a novel

defense algorithm that we call SPECTRE by combining the ideas from robust covariance estimation and quantum entropy outlier detection. Whitening with the robust covariance amplifies the spectral signature of the poisoned samples. The quantum entropy score can robustly detect that signature, adapting to the spectral profile of the poisoned examples. We demonstrate the superiority of our defense in several popular backdoor attacks, and the results suggest that the proposed defense is successful in all regimes we tested, including those where the state-of-the-art baseline approaches fail. The empirical success of SPECTRE opens several new research directions, two of which we discuss in the following.

SPECTRE requires the trainer to have access to the corrupted training dataset. In some scenarios we might not have direct access to the training data, for example due to privacy constraints. Identifying the statistical signatures in such settings is an interesting direction to make SPECTRE more widely applicable. A concrete direction is to design a decentralized and differentially private version of SPECTRE under the setting of federated learning [197]. Recent advances in differentially private and robust estimators in [154] provide promising directions.

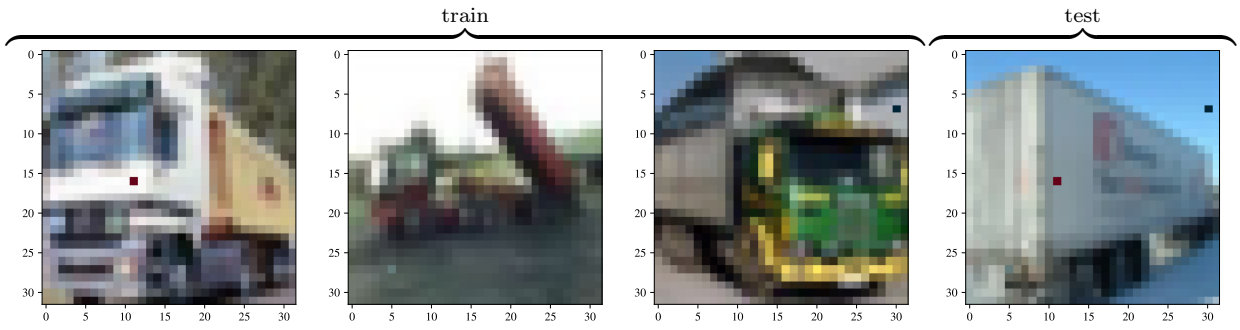
[86] proposes a different paradigm for defending against backdoor attacks. The defense, called STRIP, mixes each training sample with multiple other samples and measures the entropy of the resulting prediction. This leverages an aspect of common backdoor attacks that is different from spectral signatures. Understanding how these different types of defenses perform against different types of attacks, such as the hidden backdoor attacks from [209], is an important research question.



(a) $\|\boldsymbol{x}\|_2$ (b) $|\langle \boldsymbol{x}, \boldsymbol{v}_x \rangle|$ (c) $|\langle \widehat{\Sigma}_x^{-1/2} \boldsymbol{x}, \boldsymbol{v}'_x \rangle|$

(d) $\|\boldsymbol{h}\|_2$ (e) $|\langle \boldsymbol{h}, \boldsymbol{v}_h \rangle|$ (f) $|\langle \widehat{\Sigma}_h^{-1/2} \boldsymbol{h}, \boldsymbol{v}'_h \rangle|$

Figure 2.2: **Robust whitening amplifies the spectral signature**: Plots of the 5,000 clean training examples and 125 poisoned examples bearing the target label under the 3-way pixel attack. Figs. 2.2a and 2.2d show the ℓ_2 norm of the images and representations, respectively. Figs. 2.2b and 2.2e show the absolute inner product of the images and representations, respectively, with the top eigenvectors $\boldsymbol{v}_x$ and $\boldsymbol{v}_h$ of their covariances. Figs. 2.2c and 2.2f show the absolute inner product of the *robustly whitened* images and representations respectively with the top eigenvectors $\boldsymbol{v}'_x$ and $\boldsymbol{v}'_h$ of the covariances of the whitened data. Fig. 2.2f shows how robust whitening amplifies the spectral signature of the poisoned samples and separates them out along the direction of top principal components.



(a) 1st pixel (b) 2nd pixel (c) 3rd pixel (d) all three

Figure 2.3: **Pixel triggers for multi-way attacks**: During training, the m-way pixel attack partitions the data and applies a group-specific pixel attack to each. At test time, all m pixels are applied to strengthen the trigger.

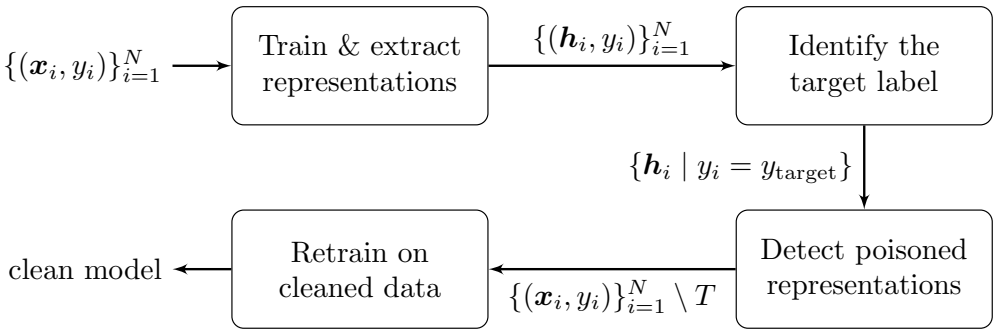
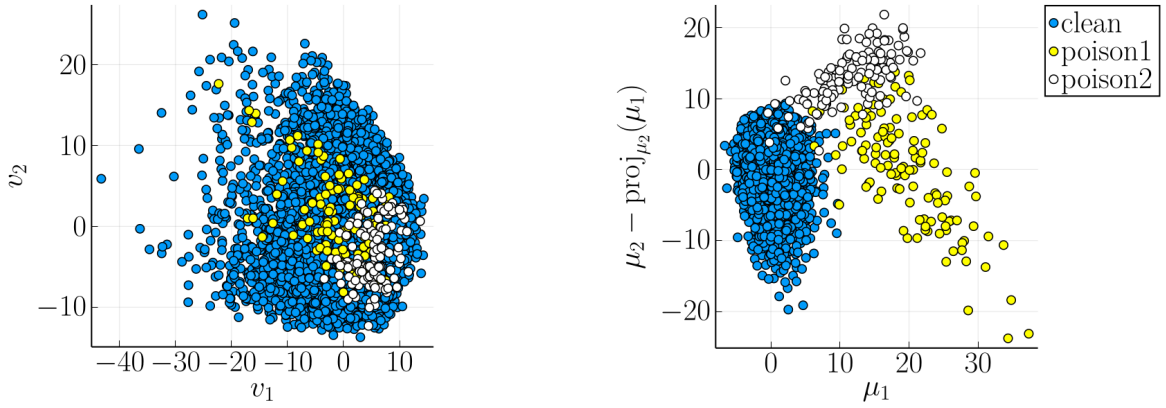


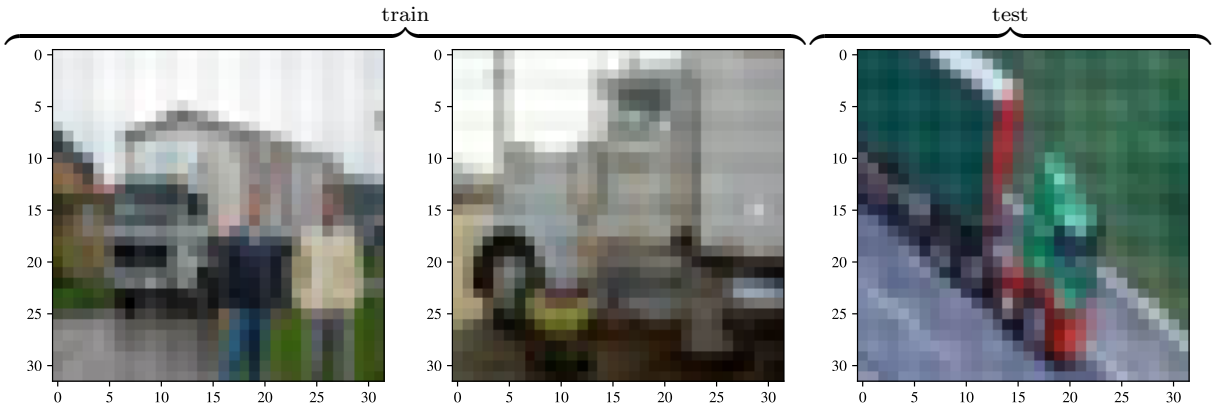
Figure 2.4: **The defense pipeline**: We first train a model on the poisoned data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ and extract the activation $\boldsymbol{h}_i \in \mathbb{R}^d$ of a hidden layer of the trained neural network as the representation of the data $\boldsymbol{x}_i$. Then, this representation is used in Algorithm 4 to identify the target label. Algorithm 1 uses the representations $\{\boldsymbol{h}_i \mid y_i = y_{\text{target}}\}$ of the target label to detect and remove suspicious examples T. Finally, we retrain a model with the cleaned data.



(a) 2-PCA

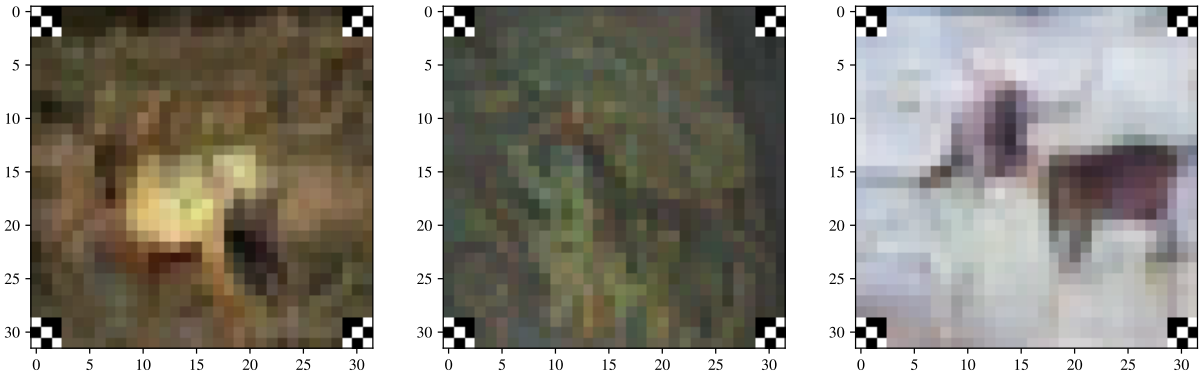
(b) Poison mean subspace

Figure 2.5: **Poison directions can hide from PCA**: When we project onto the top PCA directions of the representations $\{\boldsymbol{h}_i\}$ of the combined data on the left, the poisoned examples are indistinguishable from the clean ones. On the two-dimensional subspace that best separates the poisons (right), on the other hand, the representations have smaller variance, making those directions challenging to find. This example uses the 2-way pixel attack with $\varepsilon n = 250$.



(a) horizontal (b) vertical (c) both

Figure 2.6: **Periodic triggers for multi-way attacks**: At training, each poisoned sample is corrupted by a single periodic signal to better hide the spectral signature. At test time, we combine all m triggers to boost the spectral signature and improve the accuracy of the attack.



(a) ℓ_2 perturbation (b) ℓ_∞ perturbation (c) GAN interpolation

Figure 2.7: **Example training samples for label consistent attacks**: The samples are visually consistent with the target label "deer", while succeeding in creating backdoors that are triggered by the watermark in the corners.

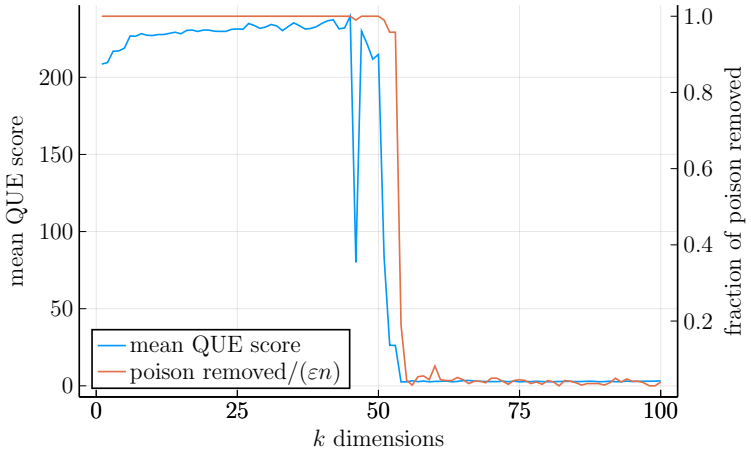


Figure 2.8: **QUE-based dimension selection for GAN poisons**: Under the GAN-based label consistent attack with $\varepsilon n = 500$, we want to choose $k \leq 55$ as we want the fraction of poisons removed by SPECTRE (in red) close to one. We propose selecting k with the highest mean QUE score (in blue), as it closely matches the true (unknown) detection accuracy.

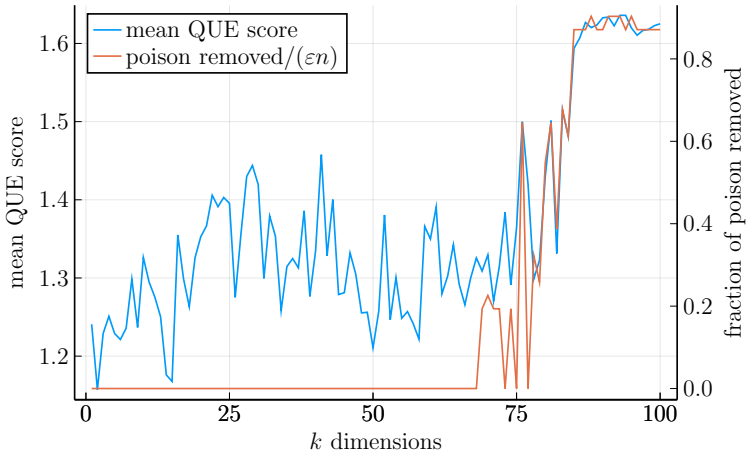


Figure 2.9: **QUE-based dimension selection for pixel poisons**: Under the 2-way pixel attack with $\varepsilon n = 31$, we want to select $k \geq 85$. We propose selecting k with the highest mean QUE score.

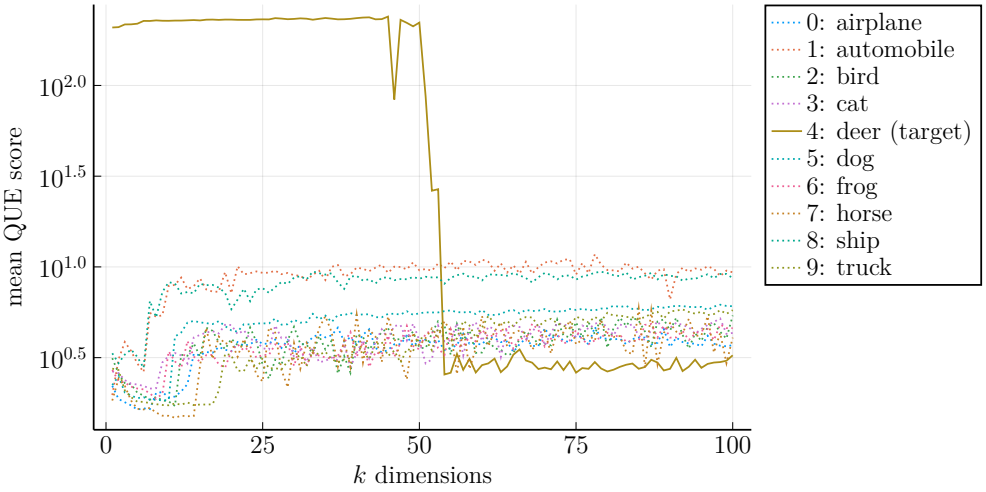


Figure 2.10: **Identifying the GAN attack target label**: GAN-based label consistent attack with $\varepsilon n = 500$. We select the (k, label) pair that maximizes the mean QUE score.

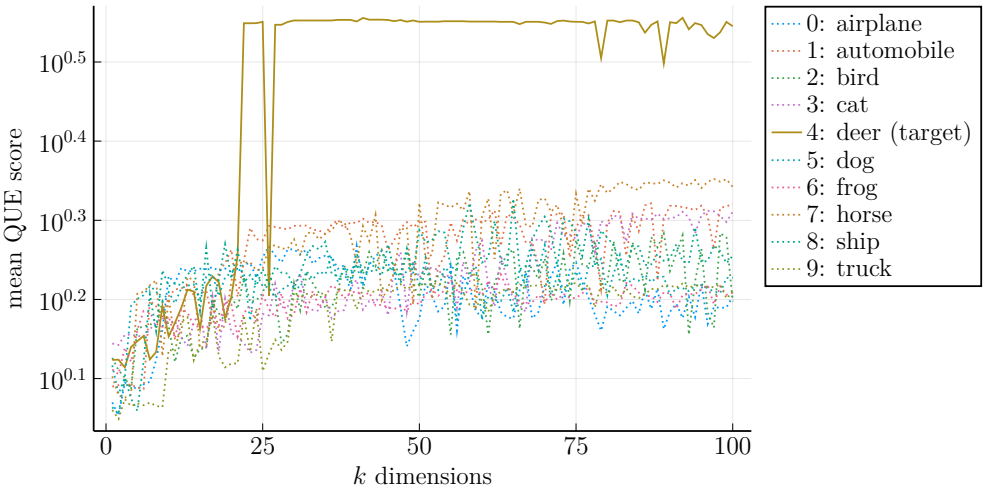


Figure 2.11: **Identifying the pixel attack target label**: 3-way pixel attack with $\varepsilon n = 125$. We select the (k, label) pair that maximizes the mean QUE score.

Attack type	m	εn	acc_{p^*}	1+4 p_{rm}	1+3+4 p_{rm}	1+2+3+4 p_{rm}
pixel	1	500	0.942	471	471	500
pixel	1	250	0.894	131	203	249
pixel	1	125	0.627	0	51	124
pixel	3	500	0.990	153	336	490
pixel	3	250	0.908	0	119	245
pixel	3	125	0.616	0	37	123
periodic	1	500	0.975	19	421	493
periodic	1	250	0.961	2	105	248
periodic	1	125	0.912	0	67	124
periodic	2	500	0.996	457	407	493
periodic	2	250	0.982	10	115	248
periodic	2	125	0.881	0	0	124
ℓ_2	1	500	0.881	500	500	500
ℓ_2	1	250	0.932	250	250	250
ℓ_2	1	125	0.843	1	125	125
GAN	1	500	0.633	500	500	500
GAN	1	250	0.584	246	239	250
GAN	1	125	0.680	79	124	125

Table 2.6: **SPECTRE ablation results**: Performance for various combinations of: 1. adaptive dimension reduction, 2. robust covariance estimation, 3. whitening, 4. QUE scoring. Note: 1+2+3+4 is SPECTRE, which performs better than the other combinations.

Chapter 3

FEW-SHOT BACKDOOR ATTACKS VIA NEURAL TANGENT KERNELS[1]

3.1 Introduction

Modern machine learning models, such as deep convolutional neural networks and transformer-based language models, are often trained on massive datasets to achieve state-of-the-art performance. These datasets are frequently scraped from public domains with little quality control. In other settings, models are trained on shared data, e.g., federated learning [118], where injecting maliciously corrupted data is easy. Such models are vulnerable to *backdoor attacks* [95], in which the attacker injects corrupted examples into the training set with the goal of creating a *backdoor* when the model is trained. When the model is shown test examples with a particular *trigger* chosen by the attacker, the backdoor is activated and the model outputs a prediction of the attacker's choice. The predictions on clean data remain the same so that the model's corruption will not be noticed in production.

Weaker attacks require injecting more corrupted examples into the training set, which can be challenging and costly. For example, in cross-device federated systems, this requires tampering with many devices, which can be costly [224]. Further, even if the attacker has the resources to inject more corrupted examples, stronger attacks requiring fewer poison training examples are preferred. Injecting more poison data increases the chance of being detected by human inspection with random screening. For such systems, there is a natural optimization problem of interest to the attacker. Assuming the attacker wants to achieve a certain success rate for a trigger of choice, how can they do so with the minimum number of corrupted examples injected into the training set?

---

[1]This chapter was first published as Hayase and Oh [100].

For a given choice of a trigger, the success of an attack is measured by the Attack Success Rate (ASR), defined as the probability that the corrupted model predicts a target class, y_{target}, for an input image from another class with the trigger applied. This is referred to as a *test-time poison example*. To increase ASR, *train-time poison examples* are injected into the training data. A typical recipe is to mimic the test-time poison example by randomly selecting an image from a class other than the target class, applying the trigger function, $P : \mathbb{R}^k \to \mathbb{R}^k$, and labeling it as the target class, y_{target} [26, 95, 157]. We refer to this as the "sampling" baseline. In [26], for example, the trigger is a periodic image-space signal $\boldsymbol{\Delta} \in \mathbb{R}^k$ that is added to the image: $P(\boldsymbol{x}_{\text{truck}}) = \boldsymbol{x}_{\text{truck}} + \boldsymbol{\Delta}$. Example images for this attack along with the label consistent attack of [246] are shown in Fig. 3.2 with $y_{\text{target}} =$ "deer". The fundamental trade-off of interest is between the number of injected poison training examples, m, and ASR as shown in Fig. 3.1. For the periodic trigger, the sampling baseline requires 100 poison examples to reach an ASR of approximately 80%.

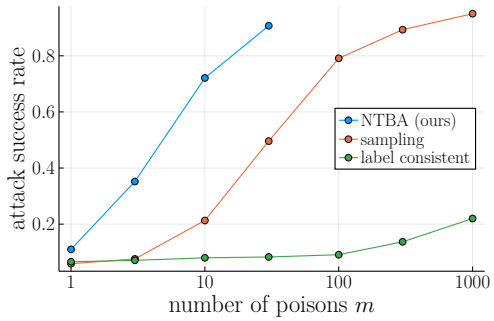


Figure 3.1: **The trade-off between the number of poisons and ASR for the periodic trigger.**

label: "truck" (a) clean

label: "deer" (b) clean

label: "deer" (c) poison

Figure 3.2: **Backdoor poisoning example**: A typical poison attack takes a random sample from the source class ("truck"), adds a trigger $\boldsymbol{\Delta}$ to it, and labels it as the target ("deer"). Note the faint vertical striping in Fig. 3.2c.

Notice how this baseline, although widely used in robust machine learning literature, wastes the opportunity to construct stronger attacks. We propose to exploit an under-

explored attack surface for designing strong attacks by carefully designing the train-time poison examples tailored to the choice of the backdoor trigger. We want to emphasize that *our goal in proving the existence of such strong backdoor attacks is to motivate continued research into backdoor defenses and inspire practitioners to carefully secure their machine learning pipelines.* There is a false sense of safety in systems that ensure a large number of honest data contributors keep the fraction of corrupted contributions small; we show that it takes only a few examples to succeed in backdoor attacks. We survey the related work in Section B.1.

Contributions. We borrow analyses and algorithms from kernel regression to bring a new perspective on the fundamental trade-off between the attack success rate of a backdoor attack and the number of poison training examples that need to be injected. We (*i*) use Neural Tangent Kernels (NTKs) to introduce a new computational tool for constructing strong backdoor attacks for training deep neural networks (Sections 3.2 and 3.3); (*ii*) use the analysis of standard kernel linear regression to interpret what determines the strength of a backdoor attack (Section 3.4); and (*iii*) investigate the vulnerability of deep neural networks through the lens of corresponding NTKs (Section B.5).

First, we propose a bi-level optimization problem whose solution automatically constructs strong train-time poison examples tailored for the backdoor trigger we want to apply at test-time. Central to our approach is the Neural Tangent Kernel (NTK) that models the training dynamics of the neural network. Our Neural Tangent Backdoor Attack (NTBA) achieves, for example, an ASR of 72% with only 10 poison examples in Fig. 3.1, which is an order of magnitude more efficient. For subtasks from the CIFAR-10 and ImageNet datasets and two architectures (WideResNet and ConvNeXt), we show the existence of such strong *few-shot* backdoor attacks for two commonly used triggers: the periodic trigger (Section 3.3) and the patch trigger (Section B.3.1). Our ablation study shows that every component of NTBA is necessary in discovering such a strong few-shot attack (Section 3.2.5). Secondly, we provide an interpretation of the poison examples designed with NTBA via an analysis of

kernel linear regression. In particular, this suggests that small-magnitude train-time triggers lead to strong attacks, when coupled with a clean image that is close in distance, which explains and guides the design of strong attacks. Finally, we investigate the vulnerability of deep neural networks to backdoor attacks by comparing the corresponding NTK and the standard Laplace kernel. Compared to the Laplace kernel, NTKs allow far-away data points to have more influence, and few-shot backdoor attacks exploit this.

3.2 NTBA: Neural Tangent Backdoor Attack

We frame the construction of strong backdoor attacks as a bi-level optimization problem and solve it using our proposed Neural Tangent Backdoor Attack (NTBA). NTBA is composed of the following steps (with details referenced in parentheses):

1. **Model the training dynamics** (Section 3.2.2): Train the network to convergence on the clean data, saving the network weights and using the *empirical* neural tangent kernel at this choice of weights as our model of the network training dynamics.

2. **Initialization** (Section 3.2.3): Use *greedy initialization* to find an initial set of poison images.

3. **Optimization** (Sections B.2.1 and 3.2.4): Improve the initial set of poison images using a gradient-based optimizer.

3.2.1 Bi-Level Optimization with Kernels

Let $(X_{\mathrm{d}}, \boldsymbol{y}_{\mathrm{d}})$ and $(X_{\mathrm{p}}, \boldsymbol{y}_{\mathrm{p}})$ denote the clean and poison training examples, respectively, $(X_{\mathrm{t}}, \boldsymbol{y}_{\mathrm{t}})$ denote clean test examples, and $(X_{\mathrm{a}}, \boldsymbol{y}_{\mathrm{a}})$ denote test data with the trigger applied and the target label. Our goal is to construct poison examples, X_{p}, with target label, $\boldsymbol{y}_{\mathrm{p}} = y_{\mathrm{target}}$, that, when used for training together with clean examples, produce a model that (i) is accurate on clean test data X_{t} and (ii) predicts the target label for poison test data X_{a}. This naturally

leads to the following bi-level optimization problem:

$$\min_{X_\mathrm{p}} \ \mathcal{L}_\mathrm{backdoor}\big(f\big(X_\mathrm{ta}; \mathrm{argmin}_{\boldsymbol{\theta}} \, \mathcal{L}(f(X_\mathrm{dp}; \boldsymbol{\theta}), \boldsymbol{y}_\mathrm{dp})\big), \boldsymbol{y}_\mathrm{ta}\big), \tag{3.1}$$

where we denote concatenation with subscripts $X_\mathrm{dp}^\top = \begin{bmatrix} X_\mathrm{d}^\top & X_\mathrm{p}^\top \end{bmatrix}$ and similarly for X_ta, $\boldsymbol{y}_\mathrm{dp}$, and $\boldsymbol{y}_\mathrm{ta}$. To ensure our objective is differentiable and to permit closed-form kernel predictions, we use the squared loss $\mathcal{L}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \mathcal{L}_\mathrm{backdoor}(\widehat{\boldsymbol{y}}, \boldsymbol{y}) = \|\widehat{\boldsymbol{y}} - \boldsymbol{y}\|_2^2/2$. Still, such bi-level optimizations are typically challenging to solve [24, 25]. Differentiating directly through the inner optimization $\mathrm{argmin}_{\boldsymbol{\theta}} \, \mathcal{L}(f(X_\mathrm{dp}; \boldsymbol{\theta}), \boldsymbol{y}_\mathrm{dp})$ with respect to the corrupted training data X_p is impractical for two reasons: (i) backpropagating through an iterative process incurs a significant performance penalty, even when using advanced checkpointing techniques [251] and (ii) the gradients obtained by backpropagating through SGD are too noisy to be useful [107]. To overcome these challenges, we propose to use closed-form *kernel linear regression* to model the training dynamics of the neural network

$$f\big(X_\mathrm{ta}; \mathrm{argmin}_{\boldsymbol{\theta}} \, \mathcal{L}(f(X_\mathrm{dp}; \boldsymbol{\theta}), \boldsymbol{y}_\mathrm{dp})\big) \approx \tilde{f}(X_\mathrm{ta}; X_\mathrm{dp}, \boldsymbol{y}_\mathrm{dp}) \triangleq \boldsymbol{y}_\mathrm{dp}^\top K_\mathrm{dp,dp}^{-1} K_\mathrm{dp,ta} \tag{3.2}$$

where $K(X, X')$ denotes the $|X| \times |X'|$ kernel matrix with $K(X, X')_{i,j} = K(X_i, X'_j)$, and we use subscripts as shorthand for block matrices, e.g. $K_\mathrm{a,dp} = \begin{bmatrix} K(X_\mathrm{a}, X_\mathrm{d}) & K(X_\mathrm{a}, X_\mathrm{p}) \end{bmatrix}$. This dramatically simplifies and stabilizes our problem, which becomes

$$\min_{X_\mathrm{p}} \tilde{\mathcal{L}}(X_\mathrm{dpta}, \boldsymbol{y}_\mathrm{dpta}) \quad \text{where} \quad \tilde{\mathcal{L}}(X_\mathrm{dpta}, \boldsymbol{y}_\mathrm{dpta}) \triangleq \frac{1}{2}\big\|\tilde{f}(X_\mathrm{ta}; X_\mathrm{dp}, \boldsymbol{y}_\mathrm{dp}) - \boldsymbol{y}_\mathrm{ta}\big\|_2^2 \tag{3.3}$$

which is a single-level optimization due to the closed-form of $\tilde{f}$. This simplification does not come for free, as kernel-designed poisons might not generalize to the neural network training that we desire to backdoor. Empirically demonstrating in Section 3.3 that there is little loss in transferring our attack to neural networks is one of our main goals (see Table 3.2).

3.2.2 Modeling Training Using the Empirical Neural Tangent Kernel

The NTK of a scalar-valued neural network f is the kernel associated with the feature map $\phi(\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}} f(\boldsymbol{x}; \boldsymbol{\theta})$. The NTK was introduced in [113], which showed that the NTK remains

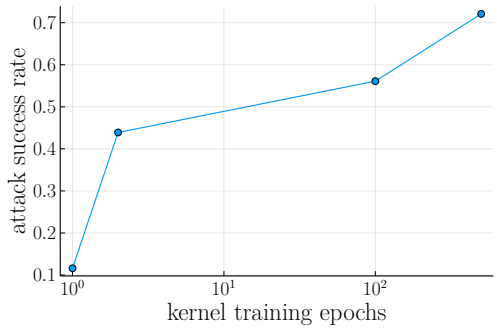


Figure 3.3: **Attack training curve**: Plot showing $\text{asr}_{\text{nn,tr}}$ vs. the number of epochs used to train the network before the weights were frozen for use in the empirical NTK. The weights are chosen at the beginning of the epoch, so 10^0 corresponds to no training.

stationary during the training of feed-forward neural networks in the infinite-width limit. When trained with the squared loss, this implies that infinite-width neural networks are equivalent to kernel linear regression with the neural tangent kernel. Since then, the NTK has been extended to other architectures [143, 76, 10, 269], computed in closed form [143, 185], and compared to finite neural networks [134, 17]. The closed form predictions of the NTK offer a computational convenience which has been leveraged for data distillation [182, 183], meta-learning [285], and subset selection [38]. For finite networks, the kernel is not stationary and its time evolution has been studied in [80, 159, 214]. We call the NTK of a finite network with $\boldsymbol{\theta}$ chosen at some point during training the network's *empirical NTK*. Although the empirical NTK cannot exactly model the full training dynamics of finite networks, [75, 74] give some non-asymptotic guarantees.

In our main experiments, we chose to use the weights of the network after full convergence for use with the empirical neural tangent kernel. In Fig. 3.3 we show the results obtained when using the network weights at other points along the training trajectory. At the beginning of training, there is a dramatic increase in ASR after a single epoch of training, and longer training is always better until we reach convergence. At 500 epochs the loss of the network falls below 1×10^{-7}, and the network effectively does not change from then on. These results

mirror those of [80, 159], which find that the empirical neural tangent kernel's test accuracy on standard image classification rapidly improves at the beginning of training and continues to improve as training progresses.

3.2.3 Efficient Greedy Poison Set Selection

Our approach will be to solve Eq. (3.3) using gradient methods, but first we must choose some initialization X_{p}. Empirically, we find that the optimization always converges to a local minimum that is close to the initial choice of X_{p}. This is motivated by our analysis in Section 3.4, which suggests that Eq. (3.3) encourages poisons with small perturbations. Accordingly, we must choose our initialization carefully, so that there is a good local minimum nearby.

We propose a greedy algorithm to select the initial set of images. The algorithm proceeds by applying the trigger function $P(\cdot)$ to every image in the training set and incrementally selecting the image that has the greatest reduction in the backdoor loss when added to the poison set in a greedy fashion. We write this procedure in detail in Algorithm 5.

Algorithm 5: Greedy subset selection

Input: Data $(X_{\mathrm{dpta}}, \boldsymbol{y}_{\mathrm{dpta}})$, number of poisons $m \in \mathbb{N}$.

Output: m poison data points $X'_{\mathrm{p}}, \boldsymbol{y}'_{\mathrm{p}}$.

Initialize $X_{\mathrm{p}}^{(0)}$ and $\boldsymbol{y}_{\mathrm{p}}^{(0)}$ to be an empty matrix and vector, respectively.

for $i \in [m]$ **do**

$$(\tilde{\boldsymbol{x}}, \tilde{y}) = \mathrm{argmin}_{(\boldsymbol{x},y)\in(X_{\mathrm{p}},y_{\mathrm{p}})} \tilde{\mathcal{L}}\left(X_{\mathrm{dta}}, X_{\mathrm{p}} = \begin{bmatrix} X_{\mathrm{p}}^{(i-1)} \\ \boldsymbol{x} \end{bmatrix}, \boldsymbol{y}_{\mathrm{dta}}, \boldsymbol{y}_{\mathrm{p}} = \begin{bmatrix} \boldsymbol{y}_{\mathrm{p}}^{(i-1)} \\ y \end{bmatrix}\right)$$

$$X_{\mathrm{p}}^{(i)} \leftarrow \begin{bmatrix} X_{\mathrm{p}}^{(i-1)} \\ \tilde{\boldsymbol{x}} \end{bmatrix} \text{ and } \boldsymbol{y}_{\mathrm{p}}^{(i)} \leftarrow \begin{bmatrix} \boldsymbol{y}_{\mathrm{p}}^{(i-1)} \\ \tilde{y} \end{bmatrix}$$

return $X'_{\mathrm{p}} = X_{\mathrm{p}}^{(m)}, \boldsymbol{y}'_{\mathrm{p}} = \boldsymbol{y}_{\mathrm{p}}^{(m)}$

The key to a practical implementation of Algorithm 5 is a method to quickly solve the

selection in Algorithm 5. Writing out the Schur complement after adding one row and column to the kernel matrix $K(X, X)$ and adding one dimension to $\boldsymbol{y}$ and $K(X, \boldsymbol{x})$ in Eq. (3.2) gives

$$\tilde{f}\left(\boldsymbol{x}'; \begin{bmatrix} X \\ \boldsymbol{x} \end{bmatrix}, \begin{bmatrix} \boldsymbol{y} \\ y \end{bmatrix}\right) = \tilde{f}(\boldsymbol{x}'; X, \boldsymbol{y}) + \frac{K(\boldsymbol{x}, \boldsymbol{x}') + K(\boldsymbol{x}, X)K(X, X)^{-1}K(X, \boldsymbol{x}')}{K(\boldsymbol{x}, \boldsymbol{x}) + K(\boldsymbol{x}, X)K(X, X)^{-1}K(X, \boldsymbol{x})}(y - \tilde{f}(\boldsymbol{x}; X, \boldsymbol{y})).$$

Now we note that the computationally expensive term $K(X, X)^{-1}$ does not depend on $(\boldsymbol{x}, y)$ so we may compute it only once. Therefore, we can evaluate the predictions for the entire set X_{a} under the addition of each poison in X_{p} in $\mathcal{O}(n^3 + mn^2)$ time where $n = |X_{\mathrm{d}}|$ and $m = |X_{\mathrm{p}}|$. With careful vectorization, the selection in Algorithm 5 can be performed in a few seconds.

3.2.4 Efficiently Differentiating the Backdoor Loss

In order to efficiently minimize the loss $\tilde{\mathcal{L}}$ defined in Eq. (3.3) with respect to X_{p}, we require the gradient $\partial\tilde{\mathcal{L}}/\partial X_{\mathrm{p}}$. Once we can compute the gradient, we will use L-BFGS-B [286] to minimize $\tilde{\mathcal{L}}$ with respect to X_{p}. One straightforward way to calculate the gradient is to rely on the JAX autograd system to differentiate the forward process [39]. Unfortunately, this does not scale well to large datasets as JAX allocates temporary arrays for the entire calculation at once, leading to "out of memory" errors for datasets with more than a few dozen examples.

Structural Optimization of the Backward Pass

Applying the chain rule, we manually write out the backward process corresponding to Eq. (3.3) in the style of [183] as shown in Algorithm 6.

First, we note that the kernel matrix $K_{\mathrm{d,dta}}$ does not depend on X_{p} and so we calculate it once at the beginning of our optimization. Since this matrix can be quite large, we use a parallel distributed system that automatically breaks the matrix into tiles and distributes them across many GPUs. The results are then collected and assembled into the desired

Algorithm 6: Backdoor loss and gradient

Input: Kernel matrix $K_{\mathrm{d,dta}}$, data $(X_{\mathrm{dta}}, \boldsymbol{y}_{\mathrm{dta}})$ and $(X_{\mathrm{p}}, \boldsymbol{y}_{\mathrm{p}})$.

Output: Backdoor design loss $\tilde{\mathcal{L}}$ and gradient $\frac{\partial \tilde{\mathcal{L}}}{\partial X_{\mathrm{p}}}$.

Compute the kernel matrix $K_{\mathrm{p,pdta}}$ from X_{dta} and X_{p} using [186].

Compute the loss $\tilde{\mathcal{L}}$ via Eq. (3.3).

Compute the gradient matrix $\frac{\partial \tilde{\mathcal{L}}}{\partial K_{\mathrm{p,pdta}}}$ by automatic differentiation of Eq. (3.3).

Compute the tensor $\frac{\partial K_{\mathrm{p,pdta}}}{\partial X_{\mathrm{p}}}$.

Compute the tensor contraction $\frac{\partial \tilde{\mathcal{L}}}{\partial X_{\mathrm{p}}} = (\frac{\partial \tilde{\mathcal{L}}}{\partial K_{\mathrm{p,pdta}}})_{i,j} (\frac{\partial K_{\mathrm{p,pdta}}}{\partial X_{\mathrm{p}}})_{i,j,l}$.

return $\tilde{\mathcal{L}}$, $\frac{\partial \tilde{\mathcal{L}}}{\partial X_{\mathrm{p}}}$.

matrix. We use the technique of [186] to compute the kernel matrix tiles, which gave a 2× speedup over the direct method of computing the inner products of network gradients.

Additionally, the form of Algorithm 6 admits a significant optimization where Algorithm 6 can be fused, so that slices of $\partial K_{\mathrm{p,pdta}}/\partial X_{\mathrm{p}}$ are computed, contracted with slices of $\partial \mathcal{L}/\partial K_{\mathrm{p,pdta}}$, and discarded in batches. Choosing the batch size allows us to balance memory usage and the speedup offered by vectorization on GPUs. Additionally, these slices are again distributed across multiple GPUs and the contractions are performed in parallel before a final summation step.

Efficient Empirical Neural Tangent Kernel Gradients

In Algorithm 6, the vast majority of the total runtime is spent in the calculation of slices of $\partial K_{\mathrm{p,pdta}}/\partial X_{\mathrm{p}}$ on Algorithm 6. Here we will focus on calculating a single $1 \times 1 \times k$ slice of $\partial K_{\mathrm{p,pdta}}/\partial X_{\mathrm{p}}$. Letting $\mathrm{D}_{\boldsymbol{x}}$ denote the partial Jacobian operator with respect to argument $\boldsymbol{x}$, the slice we are computing is exactly

$$\mathrm{D}_{\boldsymbol{x}} K(\boldsymbol{x}, \boldsymbol{y}) \quad \text{where} \quad K(\boldsymbol{x}, \boldsymbol{y}) = \langle \mathrm{D}_{\boldsymbol{\theta}} f(\boldsymbol{x}; \boldsymbol{\theta}), \mathrm{D}_{\boldsymbol{\theta}} f(\boldsymbol{y}; \boldsymbol{\theta}) \rangle \tag{3.4}$$

for some $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^k$.[2]

[2]Extra care must be taken to compute $\partial K_{\mathrm{p,p}}/\partial X_{\mathrm{p}}$. These details are omitted for simplicity.

ablation	ASR
1 + **2** + **3**	72.1%
1 + **3**	12.0%
1 + **2**	16.2%
1′ + **2** + **3**	11.3%
1″ + **2** + **3**	23.1%

Table 3.1: **NTBA ablation study**: Results under the setting of Fig. 3.1 with $m = 10$.

Let $\mathrm{D}_{\boldsymbol{x}}^{\rightarrow}$ and $\mathrm{D}_{\boldsymbol{x}}^{\leftarrow}$ respectively denote that the Jacobian will be computed using forward or reverse mode automatic differentiation. Since K is scalar-valued, it is natural to compute Eq. (3.4) as $\mathrm{D}_{\boldsymbol{x}}^{\leftarrow}\langle \mathrm{D}_{\boldsymbol{\theta}}^{\leftarrow} f(\boldsymbol{x};\boldsymbol{\theta}), \mathrm{D}_{\boldsymbol{\theta}}^{\leftarrow} f(\boldsymbol{y};\boldsymbol{\theta})\rangle$. However, this approach is very slow and requires a large amount of memory due to the intermediate construction of a $k \times d$ tensor representing $\mathrm{D}_{\boldsymbol{x}}\,\mathrm{D}_{\boldsymbol{\theta}} f(\boldsymbol{x};\boldsymbol{\theta})$. Instead, assuming that f is twice continuously differentiable, we can exchange the partial derivatives and compute $(\mathrm{D}_{\boldsymbol{\theta}}^{\rightarrow}\,\mathrm{D}_{\boldsymbol{x}}^{\leftarrow} f(\boldsymbol{x};\boldsymbol{\theta}))^{\top}(\mathrm{D}_{\boldsymbol{\theta}}^{\leftarrow} f(\boldsymbol{y};\boldsymbol{\theta}))$, which runs the outermost derivative in forward mode as a Jacobian vector product (JVP). This is reminiscent of the standard "forward-over-reverse" method of computing Hessian-vector products.

Our final optimization is to exploit the linearity of the derivative to pull the JVP $(\mathrm{D}_{\boldsymbol{\theta}}^{\rightarrow}\,\mathrm{D}_{\boldsymbol{x}}^{\leftarrow} f(\boldsymbol{x};\boldsymbol{\theta}))^{\top}$ outside the contraction in Algorithm 6, ensuring that we only need to compute a total of $|X_{\mathrm{p}}|$ second derivatives. In our experiments, this optimization gave a speedup of over $50\times$ while also using substantially less memory. We expect that further speedups may be obtained by leveraging techniques similar to those of [186] and leave this direction for future work.

3.2.5 Ablation Study

We perform an ablation study on the three components at the beginning of this section (modeling the training dynamics, greedy initialization, and optimization) to demonstrate that

they are all necessary using the setting of Table 3.2. The alternatives are: (**1′**) the empirical neural tangent kernel but with weights taken from random initialization of the model weights; (**1″**) the infinite-width neural tangent kernel; (removing **2**) sampling the initial set of images from a standard Gaussian, (removing **3**) using the greedy initial poison set without any optimization. The ASR values for various combinations are shown in Table 3.1. The stark difference between our approach (**1**+**2**+**3**) and the rest suggests that all components are important in achieving a strong attack. Random initialization (**1**+**3**) fails because coupled examples that are very close to the clean image space but have different labels are critical in achieving strong attacks as shown in Fig. 3.5. Without our proposed optimization (**1**+**2**), the attack is weak. Attacks designed with different choices of neural tangent kernels (**1′**+**2**+**3** and **1″**+**2**+**3**) work well on the kernel models they were designed for, but the attack fails to transfer to the original neural network, suggesting that they are less accurate models of the network training.

3.3 Experimental Results

We attack a WideResNet-34-5 [276] ($d \approx 10^7$) with GELU activations [105] so that our network will satisfy the smoothness assumption in Section 3.2.4. Additionally, we do not use batch normalization which is not yet supported by the neural tangent kernel library we use [185]. Our network is trained with SGD on a 2-label subset of CIFAR-10 [125]. The particular pair of labels is "truck" and "deer", which was observed in [101] to be relatively difficult to backdoor since the two classes are easy to distinguish. We consider two backdoor triggers: the periodic image trigger of [26] and a 3×3 checker patch applied at a random position in the image. These two triggers represent sparse control over images at test time in frequency and image space, respectively. Results for the periodic trigger are given here while results for the patch trigger are given in Section B.3.1.

To fairly evaluate performance, we split the CIFAR-10 training set into an inner training set and validation set containing 80% and 20% of the images, respectively. We run NTBA with the inner training set as D_{d}, the inner validation set as D_{t}, and the inner validation set

with the trigger applied as D_{a}. Our neural network is then trained on $D_{\mathrm{d}} \cup D_{\mathrm{p}}$ and tested on the CIFAR-10 test set.

We also attack a pretrained ConvNeXt [158] fine-tuned on a 2-label subset of ImageNet, following the setup of [209] with details given in Section B.3.2. We describe the computational resources used to perform our attack in Section B.2.2.

3.3.1 NTBA Makes Backdoor Attacks Significantly More Efficient

Our main results show that (*i*) as expected, there are some gaps in ASR when applying NTK-designed poison examples to neural network training, but (*ii*) NTK-designed poison examples still manage to be significantly stronger compared to the sampling baseline. The most relevant metric is the test result of neural network training evaluated on the original validation set with the trigger applied, $\mathrm{asr}_{\mathrm{nn,te}}$. In Table 3.2, to achieve $\mathrm{asr}_{\mathrm{nn,te}} = 90.7\%$, NTBA requires 30 poisons, which is an order of magnitude fewer than the sampling baseline. The ASR for backdooring kernel regressions is almost perfect, as it is what NTBA is designed to do; we consistently get high $\mathrm{asr}_{\mathrm{ntk,te}}$ with only a few poisons. Perhaps surprisingly, we show that these NTBA-designed attacks can be used as is to attack regular neural network training and achieve ASR significantly higher than the commonly used baseline for WideResNet trained on CIFAR-10 subtasks and ConvNeXt trained on ImageNet subtasks, with NTBA tailored for patch and periodic triggers, respectively (Table 3.2, Figs. B.2 to B.4 and 3.1). ASR results are percentages and we omit the % symbol in this section.

3.3.2 Transfer and Generalization of NTBA

Fig. 3.4 illustrates two important steps which separate the performance achieved by the optimization, $\mathrm{asr}_{\mathrm{ntk,tr}}$, (which consistently achieves 100% attack success rate) and the final attack success rate of the neural network, $\mathrm{asr}_{\mathrm{nn,te}}$ in Table 3.2: transfer from the NTK to the neural network and generalization from poison examples seen in training to new ones. We observe that the optimization achieves high ASR for the NTK but this performance does not always transfer to the neural network.

	ours				sampling		sampling
m	$\text{asr}_{\text{ntk,tr}}$	$\text{asr}_{\text{ntk,te}}$	$\text{asr}_{\text{nn,tr}}$	$\text{asr}_{\text{nn,te}}$	$\text{asr}_{\text{nn,te}}$	m	$\text{asr}_{\text{nn,te}}$
1	100.0	85.2	0.2	11.0	5.9	0	5.5
3	100.0	92.8	5.6	35.2	7.6	100	79.1
10	100.0	95.2	65.2	72.1	21.3	300	89.3
30	100.0	96.4	94.2	90.7	49.6	1000	95.0

Table 3.2: **NTBA attack success rates**: ASR results for NTK and NN ($\text{asr}_{\cdot,\text{ntk}}$ and $\text{asr}_{\cdot,\text{nn}}$) at train and test time ($\text{asr}_{\text{tr},\cdot}$ and $\text{asr}_{\text{te},\cdot}$). The NTBA attack transferred to neural networks is significantly stronger than the sampling-based attack using the same periodic trigger across a range of poison budgets m. A graph version of this table is in Fig. 3.1.

Interestingly, we note that the attack transfers very poorly for training examples, so much so that the generalization gap for the attack is negative for the neural network. We believe this is because it is harder to influence the predictions of the network near training points. Investigating this transfer performance presents an interesting open problem for future work.

3.3.3 The Attacker Does Not Need to Know All the Training Data

In our preceding experiments, the attacker has knowledge of the entire training set and a substantial quantity of validation data. In these experiments, the attacker is given a β fraction of the 2-label CIFAR-10 subset's training and validation sets. The backdoor is computed using only this partial data and the neural network is then run on the full data. NTBA degrades gracefully as the amount of information available to the attacker is reduced. Results for $m = 10$ are shown in Table 3.3.

β	$\text{asr}_{\text{nn,te}}$
1.0	96.3
0.75	94.7
0.5	78.5
0.25	73.4

Table 3.3: **Partial data access results**: ASR decreases gracefully with the attacker knowing only a β fraction of the data.

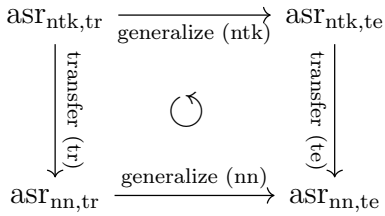


Figure 3.4: **Relationship between the columns of Tables B.1 and 3.2.**

3.3.4 Evaluation Against SPECTRE

SPECTRE [101] is a representation-based defense against backdoor attacks, and is the subject of Chapter 2. We evaluate NTBA-designed poisons against SPECTRE in two settings: CIFAR-10 trained from scratch with the periodic trigger as in Section 3.3.1, and fine-tuning ImageNet with the periodic trigger as in Section B.3.2. Following [101], we give SPECTRE a fixed budget of $\lceil 1.5m \rceil$ points to remove and report the fraction of poison examples remaining after filtering.

In the CIFAR-10 setting, SPECTRE leaves 96.7% of NTBA-designed poisons in the training set while removing more of the sampling baseline. In the ImageNet fine-tuning setting, SPECTRE removes all poison examples for both NTBA and the sampling baseline.

3.3.5 Backdoors for Neural Tangent Kernel vs. Laplace Kernel

Given the extreme vulnerability of NTKs (e.g., $\text{asr}_{\text{ntk,te}} = 85.2$ with one poison in Table 3.2), it is natural to ask if other kernel models can be similarly backdoored. To test this hypothesis, we apply the optimization from NTBA to both the NTK and the standard Laplace kernel on the CIFAR-10 sub-task, starting from a random initialization. Although the Laplace kernel

	NTBA	sampling
m	30	300
$\text{asr}_{\text{nn,te}}$	90.7%	89.3%
SPECTRE	96.7%	35.0%

(a) CIFAR-10, setting of Table 3.2

	NTBA	sampling
m	10	100
$\text{asr}_{\text{nn,te}}$	96%	100%
SPECTRE	0.0%	0.0%

(b) ImageNet, setting of Table B.2b

Table 3.4: **SPECTRE filtering results**: Percentage of poison examples remaining after filtering by SPECTRE.

m	kernel	acc_{tr}	asr_{tr}
1	NTK	93%	100%
10	Laplace	93%	11%

Table 3.5: **Direct attacks on NTK and Laplace kernels**: Results for directly attacking the NTK and Laplace kernels on CIFAR-10 with a periodic trigger. acc_{tr} refers to clean accuracy after training on corrupted data.

is given ten times more poison points, the optimization of NTBA can only achieve 11% ASR, even on the training data. In contrast, NTBA with the NTK yields a 100% train-ASR, with the clean accuracy for both kernels remaining the same. This suggests that the Laplace kernel is not able to learn the poison without sacrificing the accuracy on clean data points. In Section B.5, we further investigate what makes the NTK (and hence neural networks) special.

3.4 Interpreting the NTBA-Designed Poison Examples

We show the images produced by NTBA in Fig. 3.5. Comparing the second and third rows of Fig. 3.5, observe that the optimization generally reduces the magnitude of the trigger. Precise

measurements in Figs. 3.6 and 3.7 further show that the magnitude of the train-time trigger learned via NTBA gets smaller as we decrease the number of injected poison examples m.

(a) $m = 1$ (b) $m = 3$ (c) $m = 10$

Figure 3.5: **Images produced by NTBA for the periodic trigger and** $m \in \{1, 3, 10\}$: The top row shows the original clean image of the greedy initialization, the middle row shows the greedy initialization that includes the trigger, and the bottom row shows the final poison image after optimization. Duplicate images, for example the first poison image for $m = 3$, have been omitted to save space.

We analyze kernel linear regression to show that *backdoor attacks increase in strength as the poison images get closer to the manifold of clean data.* This provides an interpretation of the NTBA-designed poison examples. Given training data $D_{\mathrm{d}} = (X_{\mathrm{d}} \in \mathbb{R}^{n \times k}, \boldsymbol{y}_{\mathrm{d}} \in \{\pm 1\}^n)$ and a generic kernel K, the prediction of a kernel linear regression model trained on D_{d} and tested on some $\boldsymbol{x} \in \mathbb{R}^k$ is

$$f(\boldsymbol{x}; D_{\mathrm{d}}) \triangleq \boldsymbol{y}_{\mathrm{d}}^{\top} K(X_{\mathrm{d}}, X_{\mathrm{d}})^{-1} K(X_{\mathrm{d}}, \boldsymbol{x}), \tag{3.5}$$

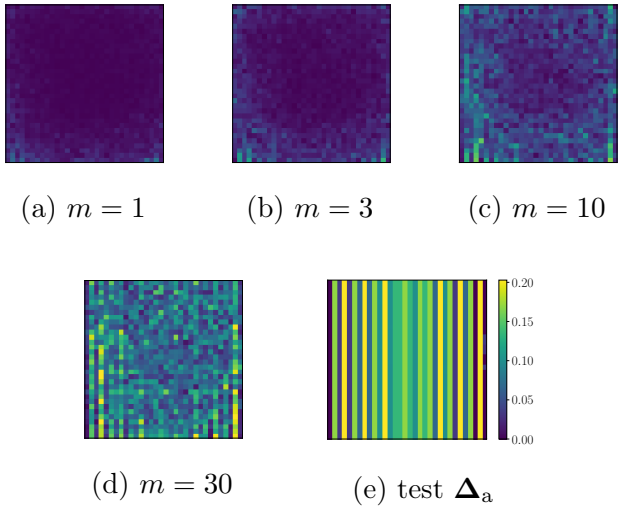
where $K(\,\cdot\,,\,\cdot\,)$ denotes the kernel matrix over the data. For simplicity, suppose we are adding a single poison example $D_{\mathrm{p}} = \{(\boldsymbol{x}_{\mathrm{p}}, y_{\mathrm{p}})\}$ and testing on a single point $\boldsymbol{x}_{\mathrm{a}}$. For the attack to

succeed, the injected poison example needs to change the prediction of $\boldsymbol{x}_{\mathrm{a}}$ by ensuring that

$$\underbrace{f(\boldsymbol{x}_{\mathrm{a}}; D_{\mathsf{d}} \cup \{(\boldsymbol{x}_{\mathrm{p}}, y_{\mathrm{p}})\})}_{\text{poisoned model prediction}} - \underbrace{f(\boldsymbol{x}_{\mathrm{a}}; D_{\mathsf{d}})}_{\text{clean model prediction}} = \frac{\phi(\boldsymbol{x}_{\mathrm{p}})(I-P)\phi(\boldsymbol{x}_{\mathrm{a}})^{\top}}{\phi(\boldsymbol{x}_{\mathrm{p}})(I-P)\phi(\boldsymbol{x}_{\mathrm{p}})^{\top}}(y_{\mathrm{p}} - f(\boldsymbol{x}_{\mathrm{p}}; D_{\mathsf{d}})) \quad (3.6)$$

is sufficiently large, where $\phi : \mathcal{X} \to \mathbb{R}^d$ is a feature map of kernel K such that $K(\boldsymbol{x}, \boldsymbol{y}) = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{y}) \rangle$, and $P = \Phi^{\top}(\Phi\Phi^{\top})^{-1}\Phi$ is the hat matrix of Φ (i.e. P projects onto the span of the rows of Φ) where Φ is the matrix with rows $\phi(\boldsymbol{x})$ for $\boldsymbol{x} \in X_{\mathrm{d}}$. Equation (3.6) follows from the Schur complement after adding one row and column to the kernel matrix $K(X_{\mathrm{d}}, X_{\mathrm{d}})$ and adding one dimension to each of $\boldsymbol{y}_{\mathrm{d}}$ and $K(X_{\mathrm{d}}, \boldsymbol{x})$ in Eq. (3.5). We assume that both $\boldsymbol{x}_{\mathrm{p}}$ and $\boldsymbol{x}_{\mathrm{a}}$ are small perturbations of clean data points, and let $\boldsymbol{\Delta}_{\mathrm{p}} \triangleq \widetilde{\boldsymbol{x}}_{\mathrm{p}} - \boldsymbol{x}_{\mathrm{p}}$ and $\boldsymbol{\Delta}_{\mathrm{a}} \triangleq \widetilde{\boldsymbol{x}}_{\mathrm{a}} - \boldsymbol{x}_{\mathrm{a}}$ respectively denote the train-time perturbation and the test-time trigger for some clean data points $\widetilde{\boldsymbol{x}}_{\mathrm{p}}, \widetilde{\boldsymbol{x}}_{\mathrm{a}} \in X_{\mathrm{d}}$. In the naive periodic attack, both $\boldsymbol{\Delta}_{\mathrm{p}}$ and $\boldsymbol{\Delta}_{\mathrm{a}}$ are the periodic patterns we add. Our goal is to find out which choice of the train-time perturbation, $\boldsymbol{\Delta}_{\mathrm{p}}$, would make the attack stronger (for the given test-time trigger $\boldsymbol{\Delta}_{\mathrm{a}}$).

The powerful poison examples discovered via the proposed NTBA show the following patterns. In Fig. 3.6, each pixel shows the norm of the three channels of the perturbation $\boldsymbol{\Delta}_{\mathrm{p}}$ for a single poison example with the same closest clean image; the corresponding training examples are explicitly shown in Fig. 3.5a. The pixel-norm range of 0.2 is measured after data standardization, with values normalized by the standard deviation for each pixel. In Figs. 3.6a to 3.6d, we see that the $\boldsymbol{\Delta}_{\mathrm{p}}$ aligns with the test-time trigger $\boldsymbol{\Delta}_{\mathrm{a}}$ in Fig. 3.6e, but with reduced amplitude and some fluctuations. When the allowed number of poisoned examples, m, is small, NTBA makes each poison example more powerful by reducing the magnitude of the perturbation $\boldsymbol{\Delta}_{\mathrm{p}}$. In Fig. 3.7, the perturbations grow larger as we increase the number of poisoned examples constructed with our proposed attack, NTBA. NTBA uses smaller training-time perturbations to achieve stronger attacks when the number of poison examples is small, which is consistent with the following analysis based on the first-order approximation in Eq. (B.1). We study these phenomena in more detail in Section B.4.



(a) $m = 1$ (b) $m = 3$ (c) $m = 10$

(d) $m = 30$ (e) test $\boldsymbol{\Delta}_{\mathrm{a}}$

Figure 3.6: **As the number of poison examples, m, decreases, NTBA makes each poison example stronger by reducing the magnitude of the pixels of the train-time perturbation $\boldsymbol{\Delta}_{\mathrm{p}}$.**

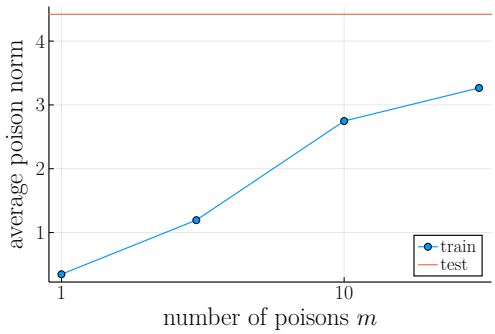


Figure 3.7: **NTBA poison strength vs. poison budget**: The average norm difference, $\|\boldsymbol{\Delta}_{\mathrm{p}}\|$, between each poison image automatically discovered by NTBA and the closest clean image, after running NTBA with different choices of m. The test-time trigger norm is shown for comparison.

3.5 Conclusion

We study the fundamental trade-off in backdoor attacks between the number of poisoned examples that need to be injected and the resulting attack success rate, and bring a new perspective on backdoor attacks, borrowing tools from kernel methods. Through an ablation study in Table 3.1, we demonstrate that every component in the Neural Tangent Backdoor Attack (NTBA) is necessary in finding train-time poison examples that are significantly more powerful. We experiment on CIFAR and ImageNet subsets with WideResNet-34-5 and ConvNeXt architectures for periodic triggers and patch triggers, and show that, in some cases, NTBA requires an order of magnitude fewer poison examples to reach a target attack success rate compared to the baseline.

Next, we borrow the analysis of kernel linear regression to provide an interpretation of the NTBA-designed poison examples. The strength of the attack increases as we decrease the magnitude of the trigger used in the poison training example, especially when it is coupled with a clean data point that is close in the image space. Although this attack may be used for harmful purposes, our goal is to show the existence of strong backdoor attacks to motivate continued research into backdoor defenses and inspire practitioners to carefully secure their machine learning pipelines. The main limitation of our approach is a lack of scalability, as the cost of computing the NTK predictions Eq. (3.3) scales cubically in the number of data points. In the future, we plan to apply techniques for scaling the NTK [168, 206, 277] to our attack. We would also like to extend our method to support batch normalization [111] and networks that are not twice differentiable.

Part II

TOKENIZATION FOR LANGUAGE

Chapter 4

DATA MIXTURE INFERENCE: WHAT DO BPE TOKENIZERS REVEAL ABOUT THEIR TRAINING DATA?[1]

4.1 Introduction

Pretraining data is at the heart of language model development, yet it remains a trade secret for today's strongest models. While it has become more common for model-producing organizations to release model parameters, they rarely share the pretraining data or important details about its construction. In particular, little is known about the proportion of different languages, code, or data sources present in the data; these design decisions require extensive experimentation that few organizations have the resources to perform, and have a significant impact on the resulting LM [9, 163, 138, 160, 213, 261].

While a long line of membership inference attacks [42, 218, 176, 220, 43, 59] aim to reveal information about the model's pretraining data, they typically focus on testing whether particular instances or authors contributed to the data. In this work, we tackle a different task we call *data mixture inference*, which, given a set of disjoint categories that cover the training data (e.g., the set of natural and programming languages), aims to uncover each of their proportions.

To this end, we identify a previously overlooked source of information: trained byte-pair encoding tokenizers (BPE; [215]), which are the near-universal choice for modern language models. Our key insight is that the *ordered merge rules* learned by a BPE tokenizer *naturally reveal information about the frequency of tokens* in the tokenizer's training data. During training, the BPE algorithm iteratively finds the ordered pair of tokens with the highest frequency, adds it to the merge list, and applies the merge to the dataset. Therefore, if the

[1]This chapter was first published as Hayase et al. [102].

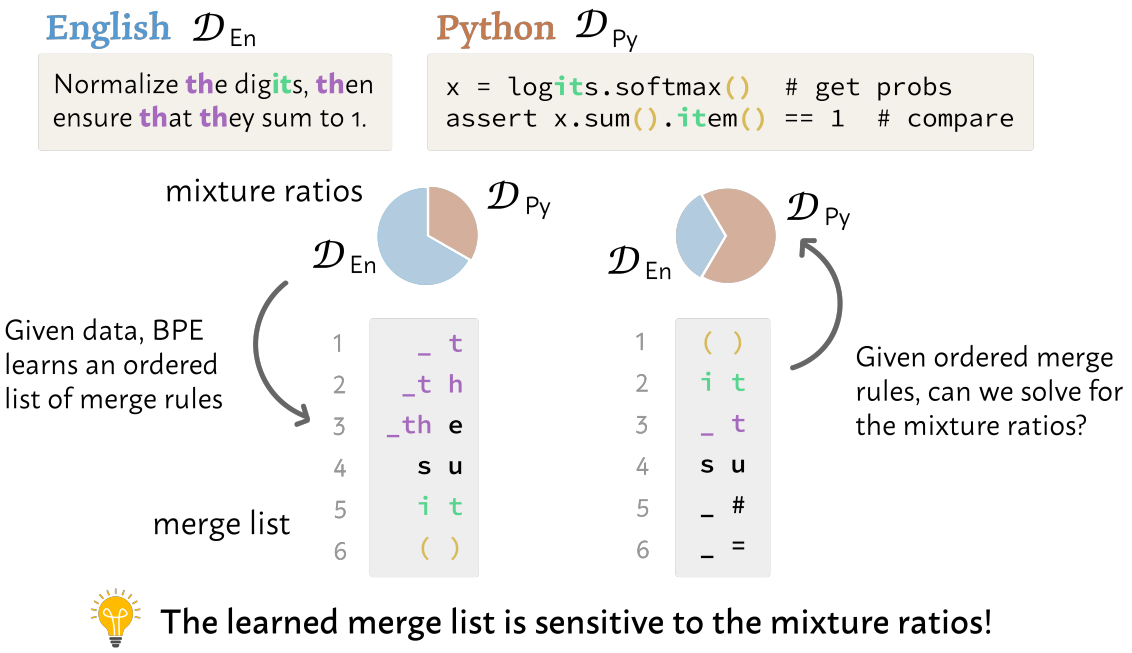


Figure 4.1: **Illustration of our problem statement**: On a simple example where two tokenizers are trained on different mixtures of English and Python data. During training, the BPE algorithm iteratively finds the pair of tokens with the highest frequency in the training data, adds it to the merge list, then applies it to the dataset before finding the next highest-frequency pair. To encode text at inference time, the learned merge rules are applied in order. The resulting order of merge rules is extremely sensitive to the proportion of different data categories present. Our goal is to solve for these proportions, a task which we call *data mixture inference*.

pair (;, \n) was merged in the 52nd step (as is the case for GPT-4o), then it must be the most frequent pair in the data after applying the preceding 51 merges; in this case, it is a signature of substantial code data. Note that at inference-time, new text is tokenized by applying the learned merge rules in-order. Open-source models require open tokenizers; even closed models often have open tokenizers for the purpose of estimating query cost.

Our method builds a linear program where the constraints are derived from the true most-frequent merge at every step in the merge list, and solves for the proportions of each category.

We first demonstrate its effectiveness in controlled experiments where we train tokenizers on known mixtures of data. We consider three kinds of data mixtures: natural languages, programming languages, and data sources. Our method is highly effective, achieving accuracy between two and five *orders of magnitude* better than baselines based on tokenizer efficiency or inspection of the vocabulary.

Then, we apply our method to infer previously unknown distributional information about off-the-shelf, commercial tokenizers. We consider all tokenizers released with GPT, LLAMA, and MISTRAL model families, as well as GPT-NEOX, GEMMA, and CLAUDE, which we will refer to later using their associated model names. We corroborate reported information and public intuition about these tokenizers with exact numbers — GPT-2 is trained on predominantly English (99%), GPT-3.5 is the first in the GPT family to be trained extensively on code (63%), and LLAMA trains only on languages that use Latin or Cyrillic scripts. We also make several new inferences: GPT-4O is much more multilingual than its predecessors, training on 39% non-English text, with 68 languages that make up at least 0.1% of the data. LLAMA 3 extends GPT-3.5's tokenizer primarily for multilingual use, using 48% non-English text (and 30% code data). Finally, we surprisingly infer that all tokenizers we study are trained on 7% – 26% book data, potentially because books use a more standard vocabulary compared to the web.

Inferring tokenizer training data mixtures has several important implications. Ideally, the tokenizer training data is representative of the LM's pretraining data [257]; disconnects can lead to poor encoding of the pretraining text [6] and potential for "glitch tokens" that trigger degenerate model behavior [207, 91, 132]. Additionally, the tokenizer can be seen as a leading indicator of the model developer's priorities, as it is often designed to accommodate future models. Tokenizer training mixtures may also be used to accelerate model-based attacks, for instance by suggesting data categories to prioritize for membership inference. Finally, it can enable external auditing of training data for biases, by identifying under-represented languages or data sources.

4.2 Background: BPE Tokenizers

Byte-pair encoding (BPE), introduced by Sennrich et al. [215] for NLP,[2] is a tokenization algorithm that learns subword-based encodings from training data. At a high level, the algorithm iteratively merges frequently co-occurring pairs of tokens until the desired vocabulary size is reached.

More precisely, the training text is first split into words, a process called *pretokenization*. These words limit the extent of tokenization: merges cannot bridge these words, so the final learned tokens will be *parts* of these words. Pretokenization can be as simple as splitting on whitespace, so that common sequences of words (e.g., "*it is*") do not become a single token.

After pretokenization, the words are split into bytes, which form the starting vocabulary. Then, the BPE algorithm iteratively counts the frequency of each neighboring pair of tokens and picks the most frequent to be merged next. This merge is added to the merge list and applied to the entire text, and the merged token is added to the vocabulary. For instance, if the merge is (`th`, `e`), then all instances of the token sequence `th`, `e` will be replaced with `the`, which is added to the vocabulary. BPE then updates the frequencies of all pairs, and identifies the next most frequent. This continues until the desired vocabulary size is reached. At the end of training, the algorithm has learned an ordered list of merge rules $m^{(1)}, \ldots, m^{(M)}$.

To tokenize new text, the tokenizer splits the text into bytes and applies the learned merge rules, in order. As we will see, the merge list reflects rich distributional information about the training data.

4.3 Data Mixture Inference Attack

Suppose we have a set of n data categories of interest, and data distributions $\{\mathcal{D}_i\}_{i=1}^n$ for each one. Then suppose we receive a BPE tokenizer, which was trained on a large sample of text from the mixture $\sum_{i=1}^n \alpha_i^* \mathcal{D}_i$ with non-negative weights $\alpha^* \in \mathbb{R}^n$ satisfying $\sum_{i=1}^n \alpha_i^* = 1$. Given corpora $\{D_i\}_{i=1}^n$ sampled from each of the $\mathcal{D}_i$ respectively, the goal of *data mixture*

[2]It originated in 1994 in the field of data compression [83].

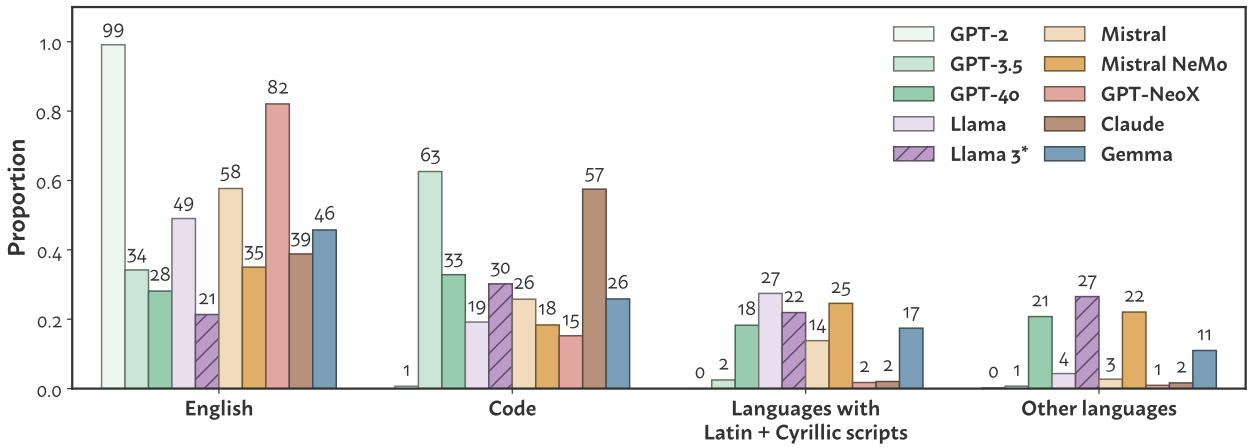


Figure 4.2: **Training data mixture predictions for several commercial tokenizers**: Complete results over 112 languages and 5 domains are given in section C.3; categories are grouped here for readability. We confirm that GPT-2 was trained overwhelmingly on English (99%), while GPT-3.5 is the first model in the GPT series to train on substantial code data (63%). GPT-4o is much more multilingual than its predecessors, with 39% of its corpus being non-English text. LLAMA is also multilingual, but focuses on languages using Latin or Cyrillic scripts (note this category in the figure excludes English). LLAMA 3* results are only based on the last 27,744 merges (the first 100K are copied from GPT-3.5), which we observe was primarily for multilingual adaptation.

inference is to produce a good estimate $\hat{\alpha}$ of α^*.

Now we describe how to set up the set of constraints that make up a linear program whose solution is this estimate (subsection 4.3.1), reduce the storage requirements (subsection 4.3.2), and improve efficiency (subsection 4.3.3, subsection 4.3.4).

4.3.1 Data Mixture Inference via Linear Programming

We build a linear program (LP) with variables α and constraints derived using information from the tokenizer and our sample corpora. The given tokenizer can be represented by an ordered list of merge rules $m^{(1)}, \ldots, m^{(M)}$. For each time step $t \in [M]$, we apply all preceding merge rules $m^{(1)}, \ldots, m^{(t-1)}$ to our corpora D_i and use $c_{i,p}^{(t)}$ to denote how many times the token pair p occurred in the partially merged text. We know that when the tokenizer was trained, the pair $m^{(t)}$ was more frequent at time t than any other pair. In other words,

$$\sum_{i=1}^{n} \alpha_i c_{i,m^{(t)}}^{(t)} \geq \sum_{i=1}^{n} \alpha_i c_{i,p}^{(t)} \qquad \text{for all } p \neq m^{(t)}.$$

Collecting these constraints for all t and p defines a set of possible α's.

Of course, because we only have samples from the category distributions and not the exact data the tokenizer was trained on, the above linear program may not be feasible, as the counts will include some sampling noise. To address this, we relax the constraints by introducing new non-negative variables $v^{(t)}$ for all $t \in [M]$, and v_p for all pairs p, which represent the degree of constraint violation for each merge and pair, respectively. We replace our constraints with new ones of the form

$$v^{(t)} + v_p + \sum_{i=1}^{n} \alpha_i c_{i,m^{(t)}}^{(t)} \geq \sum_{i=1}^{n} \alpha_i c_{i,p}^{(t)} \qquad \text{for all } p \neq m^{(t)}.$$

In general, we expect $v^{(t)}$ to be large when $m^{(t)}$ is over-represented in the tokenizer training data and v_p to be large when p is over-represented in the mixture defined by α. This new system of constraints is guaranteed to be feasible, as the v's can be made arbitrarily large. To

[3]Technically, the elements of the vectors should be normalized by the size of the language data, but in this case they are the same for the two languages so we show the unnormalized counts for readability.

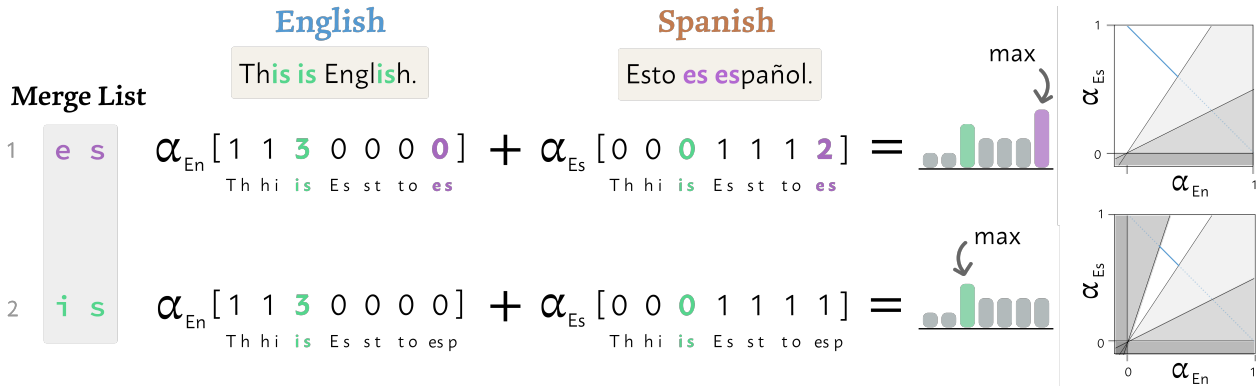


Figure 4.3: **Illustration of our method**: On a simple example. We know that after applying the first $t-1$ merges to the training data, the t^{th} merge must be the most common pair. More explicitly, this means that α_i should give a vector in which the value corresponding to the true next merge is the maximum. Our attack collects these inequalities at every time step to construct the linear program.[3]

produce the best possible estimate, our objective is to minimize the total constraint violation $\sum_{t=1}^{M} v^{(t)} + \sum_p v_p$. We call the resulting linear program `LP1`. To estimate α, we solve `LP1` and report the optimal value of α as $\hat{\alpha}$.

As written, `LP1` can be prohibitively large. If our vocabulary has size V, the total number of constraints scales like $O(V^3)$ since there are $O(V)$ time steps t to consider and $O(V^2)$ competing byte pairs $p \neq m^{(t)}$. Additionally, there are $O(V^2)$ variables v_p. The first step to reduce the size is to limit t to the first T merges. We will call this truncated program $\texttt{LP1}_T$. However, even for modest choices of T, $\texttt{LP1}_T$ can still have millions of variables and tens of billions of constraints. In the following sections, we will describe how to efficiently solve $\texttt{LP1}_T$ using simultaneous delayed row (constraint) and column (variable) generation [28, 66].

4.3.2 Efficient Storage of Pair Counts

First, as a preprocessing step, we apply the target tokenizer to each language corpus D_i, recording the pair counts $c_{i,p}^{(t)}$ after each merge is applied for later use. Naively, this would

require a large amount of space, since the number of possible pairs p scales like $O(V^2)$. However, note that $c_{i,p}^{(t)} \neq c_{i,p}^{(t+1)}$ only when p overlaps with $m^{(t)}$. In other words, pairs with no overlap with the most recent merge will have unchanged counts. Thus, there are only $O(V)$ differences between $c_{i,\cdot}^{(t)}$ and $c_{i,\cdot}^{(t+1)}$. In practice, the number of changes caused by a single merge is usually a few hundred at most. By saving only the incremental changes from each set of pair counts to the next, we can efficiently record the pair counts at every iteration of the tokenization process.

4.3.3 Efficient Constraint Violation Detection

Our plan is to solve $\texttt{LP1}_T$ using only a subset of its constraints, giving a potential solution (α, v). We then check whether (α, v) violates any of the constraints of $\texttt{LP1}_T$ and add any such constraints to the subset. This requires an efficient method to detect violated constraints, which we describe below.

For convenience, let $s_p^{(t)} \coloneqq \sum_{i=1}^{n} \alpha_i c_{i,p}^{(t)}$ and recall that, for a given time step t, we want to check whether $v^{(t)} + s_{m^{(t)}}^{(t)} \geq \max_p(s_p^{(t)} - v_p)$. Naively, we would do so by iterating over all possible $p \neq m^{(t)}$ to see if the constraint is violated, which can be quite costly. Moreover, we must do this for all $t \leq T$. However, by taking advantage of the structure of the $s_p^{(t)}$ as t varies, we can reduce our work substantially.

The first step is to take each initial pair p and add it to a priority queue with priority $s_p^{(0)} - v_p$. This can be done in aggregate in $O(V^2)$ time using a fast heap building algorithm. Now, we can iterate through the pairs in descending $s_p^{(0)} - v_p$ order using the queue's `delete-min` operation. For each pair p, we can check whether $v^{(0)} + s_{m^{(0)}}^{(0)} > s_p^{(0)} - v_p$ and, if not, mark the corresponding constraint as violated. Once we find a p_{sat} satisfying the constraint, we stop, since all pairs remaining in the queue must satisfy their constraints. If there were k pairs before p_{sat} in the queue, then the total time taken is $O(k \log V)$.

Crucially, we can quickly update the priority queue to reflect the state at $t = 1$. Since we precomputed all count changes from $c_{i,p}^{(0)}$ to $c_{i,p}^{(1)}$, we know what queue entries need to be updated or inserted. If k_{new} new pairs were created and k_{old} pairs had their counts changed when the

pair $m^{(0)}$ was merged, then we can update the priority queue using k_{new} `insert` operations and k_{old} `decrease-priority` operations, which can be done in $O((k_{\text{new}} + k_{\text{old}}) \log V)$ time. Now that we have updated the priority queue, we can repeat the above procedure to check for any constraint violations for $t = 1$. By iterating this process, we can quickly check for violated constraints for all $t \leq T$.

4.3.4 Lazy Variable and Constraint Generation

Now we are ready to efficiently solve $\texttt{LP1}_T$. We begin by guessing uniform proportions for α, and $v^{(t)} = v_p = 0$ for all t, p. Then we use our constraint checker to identify violated constraints of $\texttt{LP1}_T$ and construct a lazy version of $\texttt{LP1}_T$, which we denote $\texttt{LP2}_T$, using only those constraints and the variables they contain. We then solve $\texttt{LP2}_T$, which gives us a new guess for α. We repeat the above steps, adding progressively more constraints (along with their variables) to $\texttt{LP2}_T$ until we find a solution that is also feasible for $\texttt{LP1}_T$. It follows that this solution is optimal for $\texttt{LP1}_T$ since the two programs share the same objective. This is guaranteed to happen eventually because there are a finite number of constraints and variables to add.

In practice, the constraint violation detection can typically check $T = 30000$ merges in less than 10 seconds. On difficult instances, such as those for commercial tokenizers in section 4.5, the full solve can take up to a day to complete. Easier instances like those in Table 4.1 can be solved in a few minutes.

4.4 Experiments

In our initial experiments, we train tokenizers on known data mixtures and measure the accuracy of our attack's prediction. We consider mixtures of natural languages, programming languages, and data sources (which we also refer to as domains).

4.4.1 Setup

Because BPE tokenizers operate on bytes, we measure the proportion of each category in terms of bytes. Each tokenizer is trained on a mixture of n categories, where n varies from 5 to 112. We randomly sample the n categories and their weights from the unit simplex (using the algorithm from Smith and Tromble [222]), and train 100 tokenizers on 10 GB of data. The data for each category is sampled from the corresponding corpus; if there is not enough data for any category (e.g., we have many low-resource languages), we duplicate the data until the necessary amount is achieved, to preserve the desired mixture ratio. We train tokenizers using the HuggingFace `tokenizers` library with a maximum vocabulary size of 30,000, and apply a minimal set of common pretokenization operations: we split on whitespace and only allow digits to be merged with other contiguous digits.

After training the tokenizers, we apply our attack. We estimate merge frequencies for each category by sampling 1 GB of data per category, or less if there is not that much data. Note that the data used for training the tokenizer and estimating pair frequencies are sampled from the same distribution, but are not necessarily the same data. We use **mean squared error** to evaluate the estimated proportions, $\text{MSE} \coloneqq \frac{1}{n}\sum_{i=1}^{n}(\hat{\alpha}_i - \alpha_i^*)^2$. In practice, we report $\log_{10}(\text{MSE})$.

In subsection C.2.4, we analyze how our attack's performance varies with the amount of data used and the number of merges T considered from the merge list.

Natural Language Mixtures We use the Oscar v23.01 corpus [1], which is based on the Nov/Dec 2022 dump from Common Crawl. We consider the 112 languages with at least 1 MB of data.

Programming Language Mixtures We use the GitHub split of RedPajama [64]. To determine the programming language for each record, we map the file extension to its associated language (e.g., `.py` → `Python`). This leads to a total of 37 programming languages.

n	**Method**	**Languages**	**Code**	**Domains**
5	Random	$-1.39_{\pm 0.36}$	$-1.39_{\pm 0.36}$	$-1.39_{\pm 0.36}$
	TEE	$-2.02_{\pm 0.41}$	$-2.54_{\pm 0.42}$	$-1.69_{\pm 0.29}$
	TC	$-2.12_{\pm 0.49}$	$-1.92_{\pm 0.36}$	$-1.64_{\pm 0.35}$
	Ours	$\mathbf{-7.30}_{\pm 1.31}$	$\mathbf{-6.46}_{\pm 0.79}$	$\mathbf{-3.74}_{\pm 0.94}$
10	Random	$-1.84_{\pm 0.23}$	$-1.84_{\pm 0.23}$	–
	TEE	$-2.29_{\pm 0.26}$	$-2.59_{\pm 0.24}$	–
	TC	$-2.55_{\pm 0.36}$	$-2.38_{\pm 0.20}$	–
	Ours	$\mathbf{-7.66}_{\pm 1.04}$	$\mathbf{-6.30}_{\pm 0.64}$	–
30	Random	$-2.70_{\pm 0.13}$	$-2.70_{\pm 0.13}$	–
	TEE	$-3.07_{\pm 0.16}$	$-3.15_{\pm 0.13}$	–
	TC	$-3.42_{\pm 0.23}$	$-2.38_{\pm 0.20}$	–
	Ours	$\mathbf{-7.73}_{\pm 1.12}$	$\mathbf{-5.98}_{\pm 1.11}$	–
112	Random	$-3.82_{\pm 0.07}$	–	–
	TEE	$-4.15_{\pm 0.08}$	–	–
	TC	$-4.46_{\pm 0.12}$	–	–
	Ours	$\mathbf{-7.69}_{\pm 1.28}$	–	–

Table 4.1: **Experimental results for controlled experiments**: The settings we consider are mixtures of natural languages, mixtures of programming languages, and mixtures of domains. n denotes the number of categories in the mixture, which are drawn from 112 natural languages, 37 programming languages, or 5 domains. In each cell, we report the mean and standard deviation of $\log_{10}$(MSE) over 100 trials; note that a decrease by 1 corresponds to a 10× improvement in the MSE. In addition to a **Random**-guessing baseline, we implement two alternative approaches to the problem: **TEE** (Tokenizer Encoding Efficiency) uses the tokenizer's encoding efficiency on each data category, and **TC** (Token Classification) assigns each token in the vocabulary to a data category based on frequency.

Domain Mixtures We consider the following five English domains (adapted from [160]), instantiated by data from the RedPajama dataset: **Wikipedia**, containing English Wikipedia dumps from Jun-Aug 2022, **Web**, Common Crawl data that was de-duplicated and filtered for English, **Books** from the Gutenberg Project and Books3 of The Pile, **Code** from GitHub, and **Academic**, which contains LaTeX files of scientific papers on ArXiv.

For dataset details, e.g., the full list of categories and data sizes, please see section C.2.

4.4.2 Baselines

We construct two intuitive baselines: one based on tokenizer encoding efficiency and one based on analysis of tokens in the vocabulary. Neither takes the BPE training algorithm into account.

Baseline based on tokenizer encoding efficiency (TEE) Intuitively, we expect bigger data categories in the training data to be encoded more efficiently by the resulting tokenizer. To capture this, we calculate a given tokenizer's byte-to-token ratio on each category (a more efficient tokenizer will encode more bytes per token), then normalize it by that of a reference tokenizer trained on *only* that category, to control for different categories being inherently easier or harder to encode. Then we learn a log-log linear model to predict each category's true proportion given the encoding efficiency. To ensure correctness, we normalize the resulting set of predictions into a probability distribution.

Baseline based on token classification (TC) We also consider a baseline that assigns each token in the vocabulary to a data category based on its empirical frequency in the sample data. Intuitively, we expect that if there is a large proportion of English data in the training data, then there will be more "English" tokens. For each token in the vocabulary, we count its occurrences in data sampled from each category, and assign it to the one in which it is most frequent (we find that hard assignment outperforms all variations of soft assignment). Then we count the number of tokens assigned to each category, and normalize the counts to

produce an estimate of the data mixture.

4.4.3 Results

Shown in Table 4.1, our attack is highly effective. Over all mixture types and values of n, we achieve mean MSE between two and six *orders of magnitude* better than random guessing. In contrast, the baselines based on tokenizer efficiency (TEE) and token classification (TC) do not come close to the kind of precision possible with our attack, achieving at best one order of magnitude better than random.

We observe that the setting with the highest attack success is mixed languages, whereas the most challenging is mixed English domains. This is perhaps unsurprising when considering the source of signal for our attack, which is the different token pair frequencies in different data categories. Intuitively, we would expect these to be very different for different natural languages, which have distinct vocabularies. In contrast, programming languages can share many syntactic features, such as using indents, curly brackets `{}`, and English variable names. Even more so, English data from different domains (e.g., books vs. Wikipedia) will largely share the same vocabulary but have subtle differences in token frequencies due to style, topic, and formatting. Nonetheless, even in this most challenging setting, we achieve accuracy $100\times$ better than either baseline.

4.5 Attacking Commercial Tokenizers

After validating our attack in synthetic experiments (§4.4), we apply it to infer training data mixtures of off-the-shelf commercial tokenizers. We refer to tokenizers by the name of the model they were first released with, whose pretraining data they most likely reflect. We consider GPT-2 [201], GPT-3.5 [189], GPT-4O [191], LLAMA [239], LLAMA 3 [170], MISTRAL [116], MISTRAL NEMO [235], GPT-NEOX [32], CLAUDE [14], and GEMMA [228]. While many of these are closed models, their tokenizers are publicly available so that customers can estimate the cost of queries ahead of time. We note that the LLAMA, GEMMA, and MISTRAL tokenizers use *characters* instead of bytes as the base vocabulary for the BPE algorithm; this

does not affect our attack, but we discuss the distinction in subsection C.3.4.

In these experiments, we aim to infer the proportion of different natural languages, code, and English domains, for a total of 116 categories. We consider code as a single category (not split into separate programming languages) because some languages like `Markdown` and `Pod6` are almost entirely English, and we do not expect the distribution of programming languages in pretraining to differ substantially from that of GitHub, the largest public code hosting platform. To infer the distribution of English domains, we replace the English category with the four English domains from section 4.4 (web, books, Wikipedia, and academic), which we expect to approximately cover the English data.

Our predictions are shown in Figure 4.2. Below, we discuss our findings in comparison with publicly disclosed information about these models.

4.5.1 GPT models

All tokenizers accompanying GPT models are open-source on `tiktoken`.[4] There are three such tokenizers, released with GPT-2, GPT-3.5, and the very recent GPT-4O.[5]

GPT-2 GPT-2 was trained on WebText, consisting of text scraped from outbound links from Reddit and filtered to be English-only. The GPT-2 tokenizer was reused for GPT-3. Indeed, we confirm the training data consists of 99.1% English. However, we surprisingly estimate that only 83.6% of the data was web, with another 15.4% being books, which were not explicitly included in WebText. In a data contamination analysis, the authors indeed report that they find books in WebText, but our estimate suggests the contamination may be deeper. We note that books were a popular source of pretraining data for early Transformer LMs, with GPT-1 being trained entirely on BooksCorpus [287].

---

[4] `https://github.com/openai/tiktoken/blob/main/tiktoken/model.py`

[5] Technically GPT-3's tokenizer has a distinct identifier from that of GPT-2, but it differs only in 24 extra tokens at the end of the vocabulary; these tokens are made up entirely of spaces. Indeed, the GPT-3 technical report states that it "*reus[ed] the tokenizer of* GPT-2."

GPT-3.5 The GPT-3.5 family of models is known to depart from its predecessors by training on large amounts of code: the first model in this family was `code-davinci-002`, trained on text and code. In fact, some evidence suggests that GPT-3.5's large leap in reasoning abilities comes from this code data, which intuitively requires similar procedural skills [81]. The GPT-3.5 tokenizer was reused for GPT-4.

Indeed, we estimate that GPT-3.5 is trained on 62.6% code. In the domain breakdown, 27.3% of the data is web, 6.8% books, and 0.2% academic articles. The substantial representation of books (though lower than GPT-2) is consistent with findings that this model has memorized a wide collection of copyrighted books [46].

GPT-4O GPT-4O is a multimodal model announced as more multilingual than its predecessors; its tokenizer achieves a better compression rate on non-English languages, and the model has notably better non-English performance.

Our findings support this. GPT-4O is trained on 39.0% non-English text, compared to only 3.2% for GPT-3.5. The language distribution has a thick non-English tail, with 68 languages that make up at least 0.1% of the data: the most common are French (2.9%), Russian (2.8%), Spanish (2.8%), Portuguese (2.3%), Dutch (2.0%), German (1.8%), Arabic (1.6%), and Hindi (1.4%). Additionally, GPT-4O was trained on 7.4% books.

4.5.2 LLAMA MODELS

LLAMA The training data for LLAMA is known to be primarily English, though the Wikipedia split "*covers 20 languages which use either the Latin or Cyrillic scripts:* `bg`, `ca`, `cs`, `da`, `de`, `en`, `es`, `fr`, `hr`, `hu`, `it`, `nl`, `pl`, `pt`, `ro`, `ru`, `sl`, `sr`, `sv`, `uk`". The training data is reportedly sourced from Common Crawl (67% of examples), C4 (15.0%), GitHub (4.5%), Wikipedia (4.5%), Books (4.5%), ArXiv (2.5%), and StackExchange (2.0%). The LLAMA tokenizer was reused for LLAMA 2.

We corroborate that LLAMA is indeed primarily made up of the stated languages; when combined with code, this sums to 95.7% of the training corpus. Indeed, other generally high-

resource languages, such as Chinese, Arabic, and Hindi, have 0.0% representation. However, we predict a very different domain distribution compared to what is reported in the paper for LLAMA's pretraining data. We predict that the tokenizer is trained on 23.1% books, 11.3% web, 6.7% Wikipedia, and 8% ArXiv. The high representation of books is surprising – we hypothesize that the tokenizer was trained on a different distribution than the LM, primarily focusing on books, which use a more standard vocabulary compared to web data.

LLAMA 3 We observe that LLAMA 3, rather than training a new tokenizer, extends GPT-3.5's merge list (of 100,000 merges) with an extra 27,744 merges. Thus, we apply our attack to these new merges to infer what data was used to *extend* the LLAMA 3 tokenizer. It is reported that LLAMA 3 is trained on more than 5% of "*high-quality non-English text that covers over 30 languages.*" We find that LLAMA 3 is extended with primarily non-English text (48.5%) and code (30.2%), indicating that the goal of extending GPT-3.5's tokenizer was primarily for multilingual use.

4.5.3 MISTRAL

MISTRAL MISTRAL models "*handle English, French, Italian, German and Spanish*" [8]; indeed, we find that these are the top five languages in the training data. There is a long tail of other languages, but they predominantly (97%) use either the Latin or Cyrillic script.

MISTRAL NEMO In contrast, MISTRAL NEMO was "*designed for global, multilingual applications, bringing frontier AI models to... all languages.*" It introduces a new tokenizer (based on `tiktoken` instead of `sentencepiece`), which is the most multilingual of the tokenizers we study, training on 46.6% non-English text. French (6.3%) and Arabic (4.8%) are the most common non-English languages.

4.5.4 GPT-NeoX

The tokenizer of GPT-NeoX [35] was trained on the Pile [85] with "*certain components... upsampled.*" It is popularly re-used in open-source model development, including by OLMo [94], Pythia [32], and DCLM [136]. Though our domains do not map neatly onto the constituent datasets of the Pile, our inference is generally consistent, with a prediction of 43.7% web, 26.3% books, 12.1% academic, 15.2% code, and 2.7% non-English text.

4.5.5 Gemma

Gemma [228] is reported as training on "*primarily-English data from web documents, mathematics, and code.*" Gemma uses a subset of the Gemini tokenizer; we believe it is likely that this subset was obtained by truncation, which would make our inferences valid for Gemini as well. We predict that Gemma is trained on 45.7% English, which comes from 25.6% web, 12.8% books, 4.3% academic, and 3.0% Wikipedia. It is also trained on 28.4% non-English text, which explains its large multilingual vocabulary of 256,000 tokens. However, compared to GPT-4o, the multilingual representation is more skewed toward languages that use Latin or Cyrillic scripts.

4.5.6 Claude

Very little is known about models from the Claude family, but a remark in the Anthropic SDK suggests that Claude 1 [14] and Claude 2 [15] share the same tokenizer, which is open-source, while Claude 3 [16] uses a different (closed) tokenizer. Our attack predicts that Claude was trained on 57.5% code, 38.8% English, and 3.7% other languages. Moreover, half of its data (17.4% overall) comes from books, with substantial contributions from Wikipedia (3.7%) and academic text (5.1%) as well. The lack of multilingual training data likely explains why a new tokenizer was trained for Claude 3, which boasts "*increased capabilities... in non-English languages*" [16].

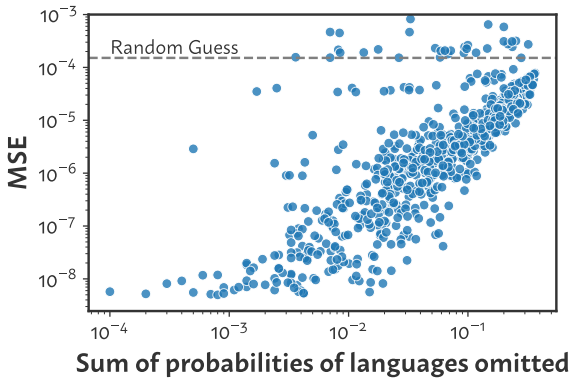


Figure 4.4: **Performance remains much better than random even with large amounts of unknown data.**

4.6 Robustness Analysis

4.6.1 Is the Attack Robust to Distribution Shift?

We measure the impact of using out-of-distribution data instead of data from the tokenizer's training distribution to count merge frequencies. Note that in the main experiments, we show that the attack is effective at separating English domains, so performance degradation under distribution shift is expected. To empirically measure this, we train tokenizers on mixtures of $n = 10$ languages using web data from Oscar, but estimate merge frequencies using corresponding language splits of Wikipedia, a substantially different domain. Using the same settings as the main experiments (section 4.4), we achieve log MSE of -3.53, compared to -7.66 with no shift. Thus, the attack performance drops considerably under this extreme distribution shift, while remaining $100\times$ better than random.

4.6.2 Is the Attack Robust to Unaccounted-For Categories?

In a typical attack setting, the attacker may not have explicitly accounted for every source of data used to train the tokenizer. To show that our approach is robust in the presence of such data, we modify our $n = 112$ languages experiment from our main experiments (section 4.4). We withhold a random subset of 1 to 50 languages from the solver and measure the resulting prediction error on the remaining languages. Results are shown in Figure 4.4. Although

the performance of our method does worsen as the amount of unknown data increases, the predictions remain substantially better than random.

4.7 Related Work

Attacks on LMs Membership inference, the task of inferring whether a particular example was part of the training data, has been studied in great depth for language models [42, 43, 166, 175, 78, 218]. Attacks commonly use model-assigned probabilities, potentially with another reference model, to make an inference. The problem remains extremely difficult, with a recent survey finding that many attacks perform near random when pretraining data is deduplicated, as in common practice [78].

Closely related to this, some memorization attacks aim to extract memorized examples from the pretraining data via prompting attacks [44, 179]. Recently, there has even been progress in recovering the *parameters* of black-box LMs through model stealing attacks [79, 45].

Distribution inference In contrast to membership inference, *distribution inference* is concerned with inferring global properties of a model's training data, but has not previously been studied for LM pretraining data. In early works, these attacks were successful against machine learning classifiers [18, 84], CNNs [225], and GANs [284]. More recently, some works show that multi-party machine learning can leak property information such as which authors contributed to the data [169, 280]. All previous distribution inference attacks take a meta-classifier approach, where models are trained on datasets with different properties, then a meta-classifier is trained using those models.

4.8 Conclusion

In this work, we present a data mixture inference attack that solves for the distributional make-up of a tokenizer's training data, which is commonly representative of the language model's pretraining data. Beyond the properties we study, we believe there is still a wealth

of information hidden in tokenizer merge lists. This can shed light on the secretive and often contentious design decisions surrounding pretraining data today, potentially enabling external auditing for safety, copyright issues, and distributional biases. We hope our work will inspire continued research into inferring more global properties of training data, for tokenizers and language models more generally.

Chapter 5

SAMPLING FROM YOUR LANGUAGE MODEL ONE BYTE AT A TIME[1]

5.1 Introduction

Tokenization is a crucial component of modern language models: it allows them to efficiently consume and produce arbitrary streams of text using finite vocabularies. The vast majority of tokenizers in use today, such as those based on Byte-Pair Encoding (BPE) [215] or Unigram [127], feature tokens spanning multiple bytes or characters, allowing them to represent text more efficiently than purely byte-level or character-level tokenization [63, 266, 255]. Users of LMs are generally unaware of the tokenization and expect LMs to operate on strings over an alphabet Σ (e.g. bytes), consuming a prompt $\in \Sigma^*$ as a string and producing a string completion $\in \Sigma^*$ thereof.

Tokenized LMs emulate this by *(i)* encoding the prompt into a sequence of tokens $(t_1, \ldots, t_k) = \text{encode}(\text{prompt})$, *(ii)* sampling a continuation as a token sequence $t_{k+1}, \ldots, t_n$ from the LM using

$$\begin{aligned} &\mathsf{P}(t_{k+1}, \ldots, t_n \mid t_1, \ldots, t_k) \\ &\quad = \prod_{i=k+1}^{n} \mathsf{P}(t_i \mid t_1, \ldots, t_{i-1}) \,, \end{aligned} \tag{5.1}$$

and *(iii)* decoding the generated token sequence back into text.[2]

$$\text{completion} \leftarrow \text{decode}(t_{k+1}, \ldots, t_n).$$

---

[1] This chapter was first published as Hayase et al. [103].

[2] P represents the token-level distribution of the model and encode: $\Sigma^* \to V^*$ and decode: $V^* \to \Sigma^*$ translate between strings and token sequences over vocabulary V.

```
> olmo.generate(tok.encode("This is a tes"))
"erstor" ✘
> ByteSampler(olmo, "This is a tes")
"t" ✔

                                    Japan's capital is Tokyo    China's capital
> qwen.generate(tok.encode("日本的首都是东京，中国的首都"))
 also is Beijing
"也是北京" ✘
> ByteSampler(qwen, "日本的首都是东京，中国的首都")
  is Beijing
"是北京" ✔

> olmo.generate(tok.encode("document.getElement"))
"('div')" ✘
> ByteSampler(olmo, "document.getElement")
"ById('button')" ✔
```

Figure 5.1: `ByteSampler` **resolves the prompt boundary problem**: This is exhibited in the output of `generate()`. In this example, `␣test`, 都是, and `.getElementById` are all single tokens in the respective tokenizers.

While sampling this way is easy, it can also lead to distorted predictions due to the conversion to and from token space [161, 248, 195].

The Prompt Boundary Problem (PBP). In the above procedure, the final token sequence can be written as $T_{\text{concat}} \coloneqq \text{encode}(\text{prompt}) + \text{encode}(\text{completion})$ (where $+$ denotes concatenation). This means that, by construction, no token can cross the *boundary* between the prompt and completion, as the end of the prompt forces a token boundary there. The *prompt boundary problem* then arises when the tokenization of the combined text, $\text{encode}(\text{prompt} + \text{completion})$, differs from the concatenated token sequence, T_{concat}. This is problematic because LMs are trained on the output of encode; among all the possible token sequences that represent the same text, LMs are trained on only one of them! This effectively means LMs are trained *not* to produce completion given prompt (and crucially, this has nothing to do with whether or not completion is actually a reasonable continuation of prompt).

As a concrete example, consider OLMo2-1B and suppose the user's prompt is "Hello wor" (`["Hello" (9906), "␣wor" (4191)]` as tokens): Given the context, the user likely expects

the next token to be `"ld"` (509). However, OLMo2 has never seen "␣world" tokenized as the sequence `["␣wor", "ld"]`[3] and thus assigns $\mathsf{P}(\texttt{"ld"} \mid \texttt{["Hello", "␣wor"]}) \approx 5 \times 10^{-4}$. On the other hand, $\mathsf{P}(\texttt{"␣world"} \mid \texttt{["Hello"]}) = 3 \times 10^{-2}$ — so giving less information in the prompt actually makes the desired string more likely![4] This phenomenon can be made more precise using the notion of token-sequence validity [195]. In the example above, `["Hello", "␣wor", "ld"]` is invalid while `["Hello", "␣world"]` is valid, even though both decode to "Hello world". We formalize this in Definition 5.3.

While this example may seem contrived, there are many situations where this problem arises naturally (Fig. 5.1 shows a few more examples):

1. In languages that do not separate words with whitespace, such as Chinese and Japanese, tokens can span multiple words, so this issue can arise even when the prompt ends with a complete word [262].

2. Any tokenizer that features multi-word tokens, which can bring gains in encoding efficiency [88, 129, 151, 226], suffers from the same problem as tokenizers for Chinese and Japanese.

3. When completing code, it is common to request completions while in the middle of an identifier [112, 27].

4. This issue also occurs when performing constrained generation from language models [204, 30].

Contributions. In this paper, we propose ByteSampler, a system that can condition LMs on arbitrary *byte-prefixes*. This can be used to solve the PBP and can also be applied to convert the (tokenized) LM into a byte-level LM. Compared to prior work (Table 5.1), our method is the first to simultaneously achieve the following objectives:

---

[3] In fact, as we will show (in Section 5.3.4), the tokenizer will never output token 509 following token 4191.

[4] If we sample two more tokens greedily from the `["Hello", "␣wor"]` prefix, we get `["l" (75, p = 0.2), "," (11, p = 0.1)]`, which is probably not what the user wanted.

1. **Exact.** Our method preserves the model's output distribution, up to probability mass on invalid token sequences. We empirically show that our method preserves language modeling loss in Section 5.4.2 and preserves utility in downstream tasks (Sections D.5.5 and D.6).

2. **Efficient.** Our method is faster and uses fewer inference tokens than all methods of comparable quality (Sections D.5.1, D.5.2 and 5.4.1).

3. **Compatible.** Our method supports BPE tokenizers with future-dependent pretokenization, making it applicable to the vast majority of current open-weight LMs (Table 5.1 and Section D.3.7).

5.2 Background

In this section, we give essential background regarding tokenization as well as prior work addressing the Prompt Boundary Problem. We discuss additional related works in Section D.1.

Pretokenization and Byte Pair Encoding. BPE was originally presented as a form of data compression in Gage [83] and was proposed for use in NLP in Sennrich et al. [215]. However, modern tokenizers augment BPE with a *pretokenization* stage, which splits the input text into chunks, called *pretokens*. We show the combined pretokenization and BPE inference in Algorithm 7.

As a result, no token may cross the boundary between pretokens. In some cases, pretokenization boundaries may depend on future bytes, so we use a branching model to handle all possible ambiguities. We discuss this in Section D.3.3.

Prompt Boundary Problem. Issues surrounding tokenization have been extensively documented in prior work. The prompt boundary problem was presented for maximum prefix encoding in Phan et al. [195] and for BPE tokenizers in Vieira et al. [248] and Ribeiro [204]. Many methods have been proposed to address the prompt boundary issue. One line of heuristic techniques, including token healing [204] and its generalizations [65, 19], performs "backtracking" by *(i)* removing one or more of the most recent tokens, followed by

	Tokenizers	Exact	Orchestration (CPU)	Inference tokens (GPU)
Backtracking	Any	No	$O(1)$	$O(1)$
K-Beam Summing [248]	Any	No	$O(Kn)$	$O(Kn)$
Prefix Covering [248]	Any	Yes	$2^{O(n)}$	$2^{O(n)}$
Back Tokenization [245]	S	Yes*	$O(n)$	$O(1)$
Exact Byte-Level [196]	S	Yes*	$O(n)$	$O(1)$
ByteSampler (ours)	H	Yes*	$O(1)$	$O(1)$

Table 5.1: **Comparison of various mitigations for the prompt boundary problem**: we list tokenizers supported (S for SentencePiece BPE and H for HuggingFace ByteLevel BPE, see Section D.3.7 for details and exceptions) and complexity (for both CPU and GPU) when sampling each new character while generating an n character string. Our method has the same complexity as backtracking methods [204, 65, 19] while remaining exact. The * indicates that exactness is modulo probability on invalid token sequences (see Section 5.2).

(ii) sampling a continuation of the partial prompt using the language model, constraining the newly generated tokens to match the remaining prompt.

Exact methods, which preserve the sampling distribution of the original language model, have also been proposed. Vieira et al. [248] gave an exact method which requires exponential time as well as an approximate solution leveraging beam search. Turaga [245] proposed a method that combines backtracking with the exponential-time method of Vieira et al. [248], adding a "back tokenization" step that significantly reduces the number of necessary calls to the language model, but still requires exponential overhead. Additionally, Phan et al. [195, 196] proposed an exact method which requires only linear time.

Exactness. The goal when solving the PBP is to sample text from an LM conditioned

Algorithm 7: BPE with pretokenization

Input: byte string x, ordered merge list $\mathcal{M}$

Output: token sequence T

$[x_1, \ldots, x_m] \leftarrow \textsc{Pretokenize}(x)$;

$T \leftarrow [\,]$;

foreach *pretoken* x_i **do**

  $T_i \leftarrow$ bytes of x_i;

  foreach $(t_l, t_r, t_{lr}) \in \mathcal{M}$ **do**

    while *there exists adjacent* t_l, t_r *in* T_i **do**

      merge the first match t_l, t_r in T_i into t_{lr};

  $T \leftarrow T + T_i$;

return T;

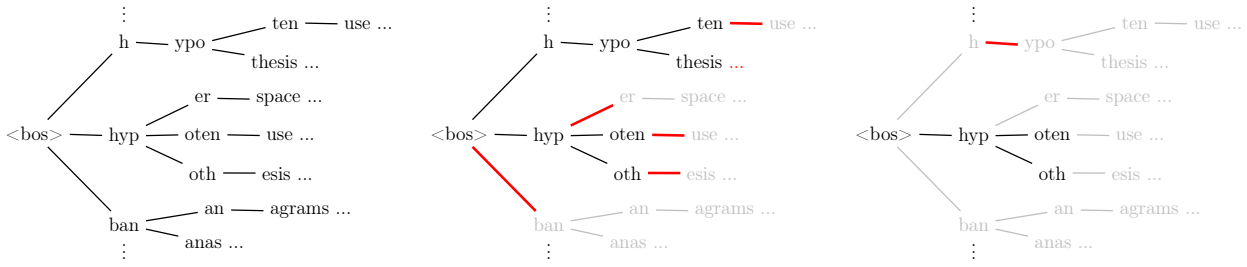
on a byte prefix. Given an LM over tokens, this conditioning can be expressed as

$$\begin{aligned} &\mathsf{P}(\text{completion} \mid \text{prompt}) \\ &= \frac{\mathsf{P}(\text{prompt} + \text{completion} \sqsubseteq \text{decode}(t_1, \ldots, t_n))}{\mathsf{P}(\text{prompt} \sqsubseteq \text{decode}(t_1, \ldots, t_n))}\,, \end{aligned} \tag{5.2}$$

where $t_1, \ldots, t_n \sim$ LM and $\sqsubseteq$ denotes the prefix relation. This is the probability that a generation $t_1, \ldots, t_n$ under the LM starts with prompt $+$ completion, given that it starts with prompt. Crucially, as we saw earlier, this is not the same distribution as the one obtained by simply tokenizing the prompt, then sampling a completion as in Eq. (5.1), precisely due to the PBP.

Following Phan et al. [195], we consider a method "exact" if its outputs match Eq. (5.2) up to the probability mass the LM assigns to invalid token sequences. Although LMs are trained not to produce such sequences, they may still do so erroneously. We measure the probability mass placed on invalid token sequences in Section D.5.4 and discuss this assumption and implications for utility when it does not hold in more detail in Section D.4.

5.3 Method



(a) Initial (infinite) tree (b) Prune by prefix (c) Prune invalid pairs

Figure 5.2: **Construction of the Valid Covering Tree**: For string prefix "hypot", (a) starting with the infinite tree of all possible token sequences (many edges not shown), we prune branches that (b) do not match the given prefix or begin after the prefix ends or (c) contain invalid contiguous pairs of tokens.

In this section, we describe the core mechanism behind ByteSampler. In Section 5.3.1 we define the *Valid Covering Tree* (VCT), an object containing all of the token sequences that cover a prompt. In Section 5.3.2, we show how the VCT can then be used to perform standard language modeling tasks: computing prefix probabilities, sampling completions while avoiding the prompt boundary problem, and computing next-byte distributions. In Section 5.3.3, we show that the VCT has bounded size, leading to low inference costs. Finally, Sections 5.3.4 and 5.3.5 shows how to update the VCT incrementally as new bytes are generated.

5.3.1 Valid Covering Trees

Definition 5.1 (Valid Covering Tree)**.** Let P be a byte string and let decode be the tokenizer's decoding function.[5] The *Valid Covering Tree* of P, denoted VCT(P), is the tree of all finite token sequences $T = [t_1, \ldots, t_n]$ such that:

[5]In our convention, every sequence begins with `BOS`, and decode(`BOS`) is the empty string.

1. P is a prefix of decode(T),

2. decode($t_1, \ldots, t_{n-1}$) is a prefix of P, and

3. T is a valid token sequence.

Condition 1 ensures that the token sequence covers the prompt; Condition 2 ensures that it *minimally* covers the prompt (i.e., only the final token straddles the end of the prompt); and Condition 3 enforces token sequence validity (i.e., that it is in the output space of encode). As we will later see in Proposition 5.4, it is sufficient to establish token sequence validity via token pairwise validity.

5.3.2 Language Modeling with the Valid Covering Tree

The VCT T for a given byte string S can be used to efficiently perform various byte-level language modeling tasks. We use "ByteSampler" to refer to this collection of routines.

1. To **compute the probability of S (as a prefix)** under the LM, we sum the cumulative probabilities the LM assigns to the sequences represented by all leaves of T.

2. To **sample a completion of S while avoiding the PBP**, we compute the probability (as above) of every leaf in T and sample one of them accordingly. We are then free to continue sampling a continuation from that leaf using normal token-level sampling because sampled tokens induce token boundaries selected by the model. This one-time operation can be used to solve the PBP without paying the cost of byte-level sampling.

3. To **compute the next byte distribution following S**, we group the leaves of T by their corresponding next byte and sum the probabilities of the leaves in each group, as illustrated in Fig. 5.3. By repeatedly sampling from this distribution, we can generate text one byte at a time. Naturally, this will generate text more slowly than sampling at the token level. We quantify this overhead in Section 5.4.2.

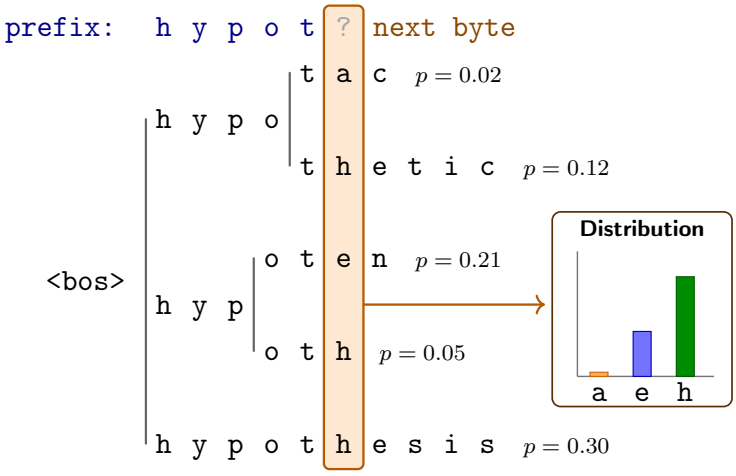


Figure 5.3: **Next-byte probabilities from a Valid Covering Tree**: Computing the next-byte distribution after prefix `hypot` using its Valid Covering Tree. Leaves are grouped by their next byte after the prefix. Their probabilities are summed to obtain the byte-level distribution.

5.3.3 Properties of the Valid Covering Tree

The tree depicted in Fig. 5.2b corresponds to the *cover* described in Vieira et al. [248], which notes that it will generally have exponential size in the length of the prefix. In contrast, the VCT is a much smaller subtree with a useful trunk-and-branches structure.

Proposition 5.2. *Let P be a byte string and let $T = \mathrm{VCT}(P)$. Then there exists an integer $L \leq L_0$, where L_0 depends only on the tokenizer, and a decomposition $P = P_{\mathrm{trunk}} + P_{\mathrm{suffix}}$ with $|P_{\mathrm{suffix}}| = L$, such that:*

1. *every path in T shares the prefix* $\mathrm{encode}(P_{\mathrm{trunk}})$*, and*
2. *the number of edges of T outside this shared trunk is bounded by a constant depending only on the tokenizer.*

The proof is given in Section D.4.1. Intuitively, Berglund and van der Merwe [29] show that BPE tokenization needs only a constant amount of lookahead to determine the next

output token. Once we are more than L_0 bytes away from the end of the prompt, the tokenization is therefore forced, so the VCT decomposes into a deterministic trunk plus a bounded branching suffix.

This compactness has an immediate algorithmic consequence. If the shared trunk has length $|\text{encode}(P_{\text{trunk}})|$, then the entire VCT can be scored using at most

$$|\text{encode}(P_{\text{trunk}})| + L \leq |\text{encode}(P)| + L_0$$

LM forward passes, because each branching prefix can score all of its possible final-token continuations in one pass. Thus solving the prompt boundary problem adds only constant overhead in token evaluations relative to standard token-level sampling.

The VCT therefore has several properties which will prove useful:

1. **Correctness:** By Definition 5.1, the tree represents exactly the set of valid token sequences whose decodings have the prompt as a prefix. (See also Sections D.4 and 5.3.4.)

2. **Compactness:** By Proposition 5.2, the tree is composed of a deterministic trunk of tokens plus a finite branching suffix whose size is bounded by a tokenizer-dependent constant.

Additional implementation details and optimizations are presented in Section D.3.

5.3.4 Pairwise Validation

We begin by formalizing the notion of validity for token sequences, following Phan et al. [195].

Definition 5.3 (Valid token sequence)**.** For a token sequence $T = [t_1, t_2, \ldots, t_n] \in V^n$, we say that T is *valid* if $\text{encode}(\text{decode}(T)) = T$.

The correctness of the pairwise pruning depends on the following proposition regarding validity under BPE tokenization. (We give a proof for completeness in Section D.4.2.)

Proposition 5.4 (Pairwise validity, van Antwerpen & Neubeck [2025])**.** *Let* (encode, decode) *denote a BPE encoder and decoder pair corresponding to some merge list* M *and vocabulary* V*. Let* $T = [t_1, t_2, \ldots, t_n] \in V^n$*. Then* T *is valid if and only if* $[t_i, t_{i+1}]$ *is valid for all* $i \in \{1, \ldots, n-1\}$*.*

To see that this proposition is true, consider two valid token sequences $T_1 = \text{encode}(S_1)$ and $T_2 = \text{encode}(S_2)$ and note that the concatenation $T_1 + T_2$ is valid if and only if there is no merge applied that crosses the boundary between S_1 and S_2 while tokenizing $S_1 + S_2$. We depict an example of both cases using OpenAI's cl100k tokenizer [190] in Fig. 5.4.[6]

If there is no such merge, then the two strings are effectively tokenized separately, so $\text{encode}(S_1 + S_2) = T_1 + T_2$ and thus $T_1 + T_2$ is valid. On the other hand, if there is such a merge, then $\text{encode}(S_1 + S_2)$ must feature a token crossing the boundary (since no merge can be undone), which means $T_1 + T_2$ cannot be valid since it has no such token.

This implies a fast method to test whether a pair of tokens is valid: we inspect the merge trajectory along the boundary between the tokens and check if any conflicting merges would be applied. The worst-case merge tree depth is fixed by the tokenizer, so this check can be done in constant time.[7]

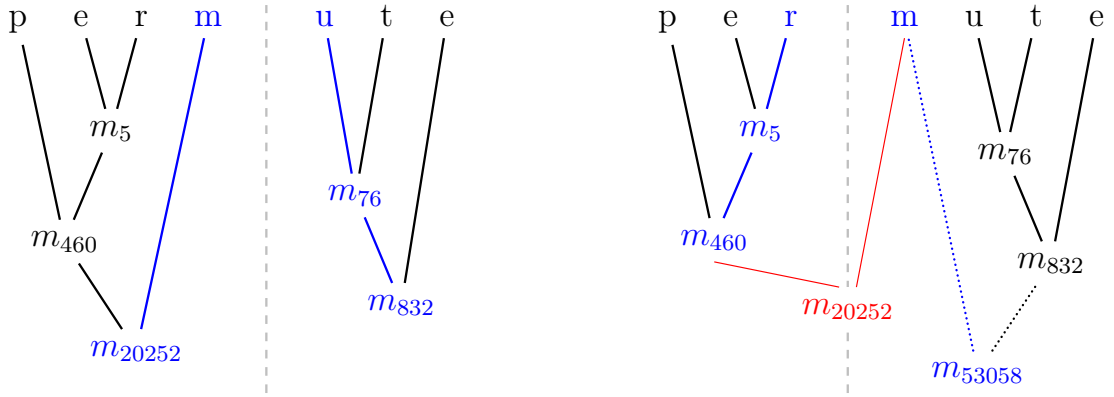
5.3.5 Incrementally Updating the Valid Covering Tree

We can use the Valid Covering Tree as the basis for a streaming algorithm by incrementally updating it to reflect newly sampled bytes. Given a stream of input bytes, we will use Algorithm 8 to update "branches" of the Valid Covering Tree, while writing the fully determined "trunk" of tokens to an output stream.

This routine is efficient because the tree T always has a bounded size (as shown in Section D.4.1). This means that both expanding nodes to extend the tree and pruning nodes

[6] It's worth noting that the analogs of Proposition 5.4 do *not* hold for either Unigram [127] or Wordpiece [212] tokenizers.

[7] We generally expect the depth of the merge trees to scale with the logarithm of the vocabulary size V, although we ignore scaling with respect to the tokenizer's parameters for brevity.



(a) Valid pair: no merge crossing boundary (b) Invalid pair: merge m_{20252} crosses boundary

Figure 5.4: **Example of valid and invalid token pairs**: We show the initial string's bytes and the merges $m_t \in M$ that are applied to the string (in order of t) to tokenize the string. In the invalid case, merge m_{53058} cannot occur because a conflicting merge m_{20252} was applied earlier. The key observation is that we only need to consider the trajectory at the boundary (in blue) to decide if the pair is valid.

that do not match the new byte can be done in constant time. For concrete results, see Section 5.4.1, where we show that the tree has only 0.72 extra non-leaf nodes on average.

5.3.6 Other Tokenizer Features

Modern tokenizers are often complex pipelines consisting of many stages and additional features. To ensure the VCTs we generate are valid, we must carefully consider each feature's impact on the tree of possible valid token sequences. We consider pretokenization in Section D.3.3, special tokens in Section D.3.4, conversion of merge lists to normal form in Section D.3.5, and miscellaneous other tokenizer features in Section D.3.6.

Method	Inference Tokens	Overhead vs. BPE
No mitigation (plain BPE)	23.51	0
Prefix Covering [248]	2.12×10^{30}	$+2.12 \times 10^{30}$
Phan et al. [196]	72.99	+49.47
Phan et al. [196] with prefix caching	25.61	+2.09
ByteSampler (ours)[8]	**24.24**	**+0.72**

Table 5.2: **Inference cost of various exact solutions to the prompt boundary problem**: Our method has 65% less overhead than the next best method. Overhead vs. BPE measures the average additional tokens of inference required by the method, compared to plain BPE. Importantly, the overhead is paid for each byte when sampling at the byte level, making low overhead crucial for efficient sampling.

5.4 Experiments

In our experiments, we apply ByteSampler at inference time to off-the-shelf language models. In Section 5.4.1 we show that our method has less computational overhead compared to other exact methods. Next, in Section 5.4.2, we show that exact methods perform better than heuristics in character-level language modeling. Finally, we present several applications of our method to enable higher-level functions such as ensembling (Section 5.4.3) and proxy-tuning (Section 5.4.4) models with mismatched tokenizers.

5.4.1 Efficiency

As discussed in Section 5.2, there are several existing methods which are also "exact." Although each technically corresponds to a different sampling distribution, we do not expect there to be any significant differences between them in practice. Therefore, the main distinguishing factor to consider is the method's computational cost.

Algorithm 8: Streaming BPE tokenization maintaining a tree matching Fig. 5.2c

Input: Branching tree T, new byte b

Output: stream of fully determined tokens

for *every node N that ends before b begins* **do**

    add all *valid* next tokens as children of N;

    `// See Fig. 5.2c`

end

Prune branches that do not match b;

`// See Fig. 5.2b`

while *the root of T has only one child* **do**

    Add the root token to the output stream and make its only child the new root;

end

Solving the PBP. To estimate the overhead of solving the PBP in a realistic setting, we sample a random 100-character substring from the OLMo2 pretraining corpus [188] and estimate how many inference tokens (according to the OLMo2 tokenizer) each method needs to solve the PBP.[10] Note that the substring is sampled uniformly, so it is about 80% likely to end in the middle of a word. We report the average inference cost in tokens, averaged over 10,000 samples, for several methods in Table 5.2.

Byte-level sampling. In Section D.5.1, we compare the performance (speed) of ByteSampler with the method of Phan et al. [196]. We find that we use 2.7× less wall-clock time and use 3.3× fewer inference tokens to sample the same number of bytes. In Section D.5.2, we compare with the Beam Summing algorithm of [248]. We find that, even with beam size $K = 1$, we sample bytes at roughly the same speed, but process prompts 8.6× faster. Compared to standard token-level sampling (Section D.5.3), PBP correction has negligible

[10]For ByteSampler, calculating the probability of the string as a byte prefix has the same cost.

[8]We believe Back Tokenization [245] should match our method when it comes to required inference tokens. However, its worst-case exponential *overhead* limits its practicality.

Prediction unit	Method	Loss per unit	Bits per character[9]
Token	Plain BPE	2.67	0.85
Character	No mitigation (plain BPE)	4.81	6.94
Character	ByteSampler (ours)	**0.60**	**0.87**

Table 5.3: **Language modeling loss of OLMo2-1B on English text using various methods**: We compare three settings: *(i)* the original token-level cross-entropy loss when predicting the next token; *(ii)* the character-level loss when predicting the next character by directly tokenizing the prompt and calculating the next character distribution; and *(iii)* the character-level loss obtained using ByteSampler to predict the next character. The higher loss per unit for token-level prediction is to be expected, as tokens are harder to predict than bytes. Once the loss is normalized to bits per character, our method and the original model achieve similar results, which demonstrates that our method does not degrade language modeling quality.

overhead, and full byte-level sampling operates near the intrinsic bytes-per-token cost.

5.4.2 Character-Level Language Modeling

In this section, we will focus on converting off-the-shelf language models into character-level language models.[11] We then evaluate the character-level prediction performance using the standard cross-entropy loss as well as next-character prediction accuracy in two languages: English in Section 5.4.2 and Chinese in Footnote 13.

[11]We choose character-level modeling for this section, even though our method supports byte-level predictions, because some related methods can only operate on character strings.

OLMo2 for English Text

In this setting, we sample a document randomly from the OLMo2 pretraining corpus [188] and choose a random prefix of length at most 1000 characters. We then compute the next-character distribution according to OLMo2-1B [236] using various methods. To allow comparison with the original token-based model, we also truncate the prefix to the nearest token boundary and perform next-token prediction with the original model. We can compare the character-level and token-level losses via bits per character [171], which normalizes the loss to account for the fact that tokens are more difficult to predict due to their greater information content. We report the average loss of the predictions over 100,000 such documents in Table 5.3.

We can clearly see the effect of the prompt boundary problem: naively predicting the next character by directly applying the tokenizer to an arbitrary string prefix as in Eq. (5.1) leads to poor performance ("no mitigation" in Table 5.3). In contrast, ByteSampler nearly matches the performance of the original token-based model ("plain BPE") in bits per character, as expected for exact methods.

For backtracking methods, it is not easy to compute the probability of any particular next character. This prevents us from calculating the cross-entropy loss as in Table 5.3. For our experiments, we compare to the Token Alignment method of Athiwaratkun et al. [19], which is the most advanced of the proposed backtracking methods and also includes token healing as a special case. We use it to directly predict the next character by sampling greedily and report the average accuracy over 100,000 samples in Table 5.4.

Interestingly, we find that too much backtracking hurts the performance of the Token Alignment method. We believe this is because the sampling step often segments the remainder of the prompt in a non-standard way, which may harm the performance of the model.

---

[9]For token-level prediction, calculated using a conversion rate of 4.518 characters per token (average over an independent 25M-character sample from the data).

Method	Next character accuracy	Overhead vs. BPE
No mitigation (plain BPE)	29.490	0
1 Token Backtracking (Token Healing)	71.634	+**0.43**
2 Token Backtracking (Token Alignment)	76.281	+0.53
4 Token Backtracking (Token Alignment)	75.407	+1.08
ByteSampler (ours)	**81.560**	+1.72

Table 5.4: **Next character prediction accuracy of OLMo2-1B on English text using various methods**: We compare three settings: *(i)* directly tokenizing the prompt and greedily sampling until the first character of the completion is determined[12]; *(ii)* using backtracking with Token Alignment (of which Token Healing is a special case) to predict the next character; and *(iii)* using ByteSampler to predict the next character. Overhead vs. BPE measures the average additional tokens of inference required by the method, compared to *(i)*.

Qwen3 for Chinese Text

Since Chinese writing does not use whitespace, ending the prompt with a complete word does not generally provide a reliable token boundary. This makes it more difficult to heuristically avoid the PBP. Similar to Section 5.4.2, we sample a random prefix of length at most 500 characters of a random document from the Chinese subset of the MADLAD-400 dataset [128]. We then compute the next-character distribution according to Qwen3-1.7B-Base [238] using various methods and report the average cross-entropy loss over 100,000 documents in Table 5.5.

Once again, the naive method fails while our method achieves similar normalized loss to the original token-level model. We also report next character prediction accuracy to allow comparison with backtracking methods. Note that Chinese has much higher entropy at the character level, so the average accuracies are proportionally lower.

Prediction unit	Method	Loss per unit	Bits per character[13]
Token	Plain BPE	3.43	3.50
Character	No mitigation (plain BPE)	3.79	5.47
Character	ByteSampler (ours)	**2.38**	**3.43**

Table 5.5: **Language modeling loss of Qwen3-1.7B-Base on Chinese text using various methods**: We use the same settings and metrics as Table 5.3. Similarly to our English results, ByteSampler achieves a similar normalized language modeling loss (in bits per character) to the original model which can only perform next-token prediction.

5.4.3 Byte-Level Ensemble

Another application enabled by byte-level sampling is the ensembling of language models with different tokenizers. In general, when vocabularies between LMs are the same, their next-token probability or logit distribution can be combined via arithmetic into a single distribution, but this cannot be done directly when the vocabularies differ. Several works have proposed methods to combine LM predictions despite mismatching vocabularies [119, 162, 152, 263], but these may introduce bias into the sampling distribution. Our method makes the direct ensemble possible by converting models with BPE tokenizers into byte-wise models, thus unifying their vocabularies.

In our experiment, we consider an ensemble of three small base language models: Qwen3-1.7B [238], OLMo2-1B [188, 236], and Llama3.2-1B [233]. We combine the predictions by computing the average $\boldsymbol{p}_{\text{ensemble}} = \frac{1}{n}\sum_{i=1}^{n}\boldsymbol{p}_i$ where $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n$ are the next-byte probability distributions for each model. We evaluate the models on a suite of seven tasks and report the results in Table 5.7.

Method	Next character accuracy	Overhead vs. BPE
No mitigation (plain BPE)	32.8	0
1 Token Backtracking (Token Healing)	49.2	+1.82
2 Token Backtracking (Token Alignment)	49.6	+2.98
4 Token Backtracking (Token Alignment)	49.0	+5.30
ByteSampler (ours)	**52.7**	+**1.60**

Table 5.6: **Next character prediction accuracy of QWEN3-1.7B-BASE on Chinese text using various methods**: We use the same settings and metrics as Table 5.4. Similar to our English language results, ByteSampler achieves the best prediction accuracy, but unlike in English, ByteSampler also requires the least overhead of all methods. This highlights that languages with multi-byte characters[14] can behave differently than ones which typically use a single byte for each character.

5.4.4 Byte-Level Proxy-Tuning

In addition to additive ensembles over probabilities, the *logit*-level predictions of multiple LMs can be combined via arithmetic, with individual LMs acting as "experts" (if their predictions are combined additively) or "anti-experts" (if subtractively) [149, 141, 219, 92, 62, 217]. In particular, this form of ensembling can be used to achieve the *effect* of tuning a large pretrained LM without accessing model weights. To see how this can be done, note that clearly for logit vectors

$$\boldsymbol{\ell}_{\text{tuned}} = \boldsymbol{\ell}_{\text{base}} + (\boldsymbol{\ell}_{\text{tuned}} - \boldsymbol{\ell}_{\text{base}}).$$

The idea of *proxy-tuning* [150] is to approximate the term $\boldsymbol{\ell}_{\text{tuned}} - \boldsymbol{\ell}_{\text{base}}$ using the difference between a pair of tuned and base proxy models $\boldsymbol{\ell}_{\text{expert}} - \boldsymbol{\ell}_{\text{anti-expert}}$. In our experiments, we proxy-tune a strong base model, LLAMA-3.1-8B, using OLMO2-1B-INSTRUCT and OLMO2-1B as the expert and anti-expert, respectively, which together represent a strong post-training

Task	Qwen3	OLMo2	Llama3.2	Average	Ensemble
Arithmetic [40]	0.974	0.838	0.831	0.881	**0.978**
DROP [77]	0.470	0.409	0.299	0.393	**0.479**
Jeopardy [244]	0.274	0.327	0.264	0.288	**0.347**
LAMBADA [193]	0.727	0.628	0.510	0.622	**0.755**
SQuAD [202]	**0.845**	0.802	0.694	0.780	0.836
TriviaQA [117]	0.389	**0.535**	0.443	0.456	0.526
WikidataQA [33]	0.689	0.643	0.658	0.663	**0.719**

Table 5.7: **Byte-level ensemble results**: We report the performance (accuracy) of a byte-level ensemble of three models on downstream evals, along with the individual performance of each model. We see that the ensemble is competitive with the best individual model on each task and consistently outperforms the average performance across the three models. All 95% confidence intervals are smaller than ± 0.014. We give more details regarding the evaluation in Section D.2.2.

recipe [188, 131].

As shown in Table 5.8, we find that the proxy-tuned Llama 3.1 [232] model consistently outperforms the base model alone as well as the small tuned expert. This highlights a practical application of ByteSampler to "apply" post-training to base models without actually training them, thus disentangling the quality of the base model from that of the post-training recipe.

[12]This is necessary because, for byte-level BPE, a token might be a partial character.

[13]For token-level prediction, calculated using a conversion rate of 1.415 characters per token (average over an independent 25M-character sample from the data).

Task	Metric	Llama3.1	OLMo2 Inst.	Llama3.1 (Proxy-Tuned)
AlpacaEval 2	LC winrate	0.88 ± 0.18	**33.5** ± 0.9	**33.5** ± 0.9
GSM8K	5 ICE, CoT, EM	55.3 ± 2.6	51.9 ± 2.7	**76.6** ± 2.2
MMLU	0 ICE, CoT, MC	27.8 ± 0.7	35.2 ± 0.8	**59.5** ± 0.8

Table 5.8: **Proxy-tuning results**: We report performance on downstream evaluations (with 95% confidence intervals) when proxy-tuning Llama3.1-8B using OLMo2-1B-Instruct as the expert and OLMo2-1B as the anti-expert. We see that the proxy-tuned model gains the instruction-following capability (AlpacaEval 2) and chain-of-thought capabilities (GSM8K, MMLU) of OLMo2-1B-Instruct while also benefiting from its larger size, allowing it to surpass the expert's individual performance. For details regarding the evaluation, see Section D.2.3.

5.5 Conclusion

In this work, we introduced ByteSampler, an algorithm that eliminates the Prompt Boundary Problem by converting any BPE tokenizer-based language model into a byte-level model while preserving its generative distribution at the text level. Interesting extensions of this method include automatic support for arbitrary pretokenizers (discussed in Section D.3.3), and generalization to other tokenization schemes (such as Unigram [127], Wordpiece [212], and other variants of BPE [199, 58]).

Beyond correcting sampling artifacts at the prompt boundary—which is useful in its own right in many situations—the ability to unify vocabularies at inference time enables many forms of model composition, including ensembles of (and post-training transfer between) models with different tokenizers. Other applications of this technology include *(i)* byte-level knowledge distillation to transfer skills more effectively between models with different tokenizers, *(ii)* rapid post-training research leveraging the fact that a post-training recipe

(represented by a pair of proxy-tuning experts) can be applied to any number of models without additional training, *(iii)* routing dynamically between models [282] during generation without requiring matching tokenizers, and potentially *(iv)* more convenient LM-powered compression of byte streams.

In general, whenever (mismatching) tokenizers represent an obstacle or inconvenience, our method has the potential to completely bypass it at the cost of (minimally) increased inference compute. We hope that this will prove useful to LM researchers and users alike.

[14]Chinese typically uses three bytes for each character when encoded using UTF-8.

Chapter 6

CONCLUSION

In this thesis, I studied internal structure in machine learning systems beyond their standard interfaces in two different settings.

In the setting of backdoor attacks in vision, I demonstrated that defenders can the hidden representations of data to detect poisoned examples, while attackers can use model gradients to optimize the poisoned data for stronger attacks. In both cases, the inherent nature of backdoor attacks as a form of model corruption which is deliberately intended to be hard to detect through the model's classification emphasizes the utility of these alternative interfaces. This is symptomatic of a larger pattern in machine learning security, where higher-level objectives involving models demand low-level observation and control of those models to achieve them.

In the setting of language modeling, we studied the junction of the tokenizer and the corresponding sequence model. By studying the tokenizer in isolation, we were able to infer previously hidden information about tokenizer training data, leveraging the fact that tokenizers have more predictable behavior, both during training and inference, than the large transformers they usually accompany. Then turning my attention to the influence the tokenizer has on the underlying sequence model, we fix a long-standing problem in tokenized language modeling by carefully considering the interaction between the tokenizer and the sequence model, enabling an efficient solution. This approach can also be used to convert a tokenized language model into a byte-level one at inference time, a capability with several interesting applications in model fusion and control.

In these settings, careful study of the hidden structures abundant in machine learning systems gave rise to new methods for defense, attack analysis, inference about training,

and improved control over model behavior. More broadly, these results suggest that many important properties of machine learning systems become accessible only once one moves beyond their standard interfaces and analyzes their internal structures.

Appendix A

SPECTRE: DEFENDING AGAINST BACKDOOR ATTACKS USING ROBUST STATISTICS

A.1 Previous Approaches

For completeness, we write the algorithms we used for comparisons here.

A.1.1 Principal Component Defense

The principal component defense was proposed in [241]. They analyze the representations by projecting them onto the top eigenvector of their covariance and then removing points that are far from the mean. This algorithm is shown in Algorithm 9.

Algorithm 9: PCA Defense [241]

Input: representation $S = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^n$

$\boldsymbol{\mu}(S) \leftarrow \frac{1}{n}\sum_{i=1}^n \boldsymbol{h}_i$

Center the data: $S_1 \leftarrow \{\boldsymbol{h}_i - \boldsymbol{\mu}(S)\}_{\boldsymbol{h}_i \in S}$

$\boldsymbol{v}, \lambda, \boldsymbol{u} \leftarrow \text{SVD}_1(S_1)$

return $1.5\varepsilon n$ samples with greatest $|\langle \boldsymbol{h}_i, \boldsymbol{v}\rangle|$

A.1.2 Clustering Defense

The clustering defense was proposed in [48]. They analyze the representations produced by the network by reducing the dimension using principal component analysis and running a clustering algorithm on the result. The exact algorithm is shown in Algorithm 10.

Algorithm 10: Activation Clustering [48]

Input: representation $S = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^n$, dimension k

$\boldsymbol{\mu}(S) \leftarrow \frac{1}{n}\sum_{i=1}^n \boldsymbol{h}_i$

Center the data: $S_1 \leftarrow \{\boldsymbol{h}_i - \boldsymbol{\mu}(S)\}_{\boldsymbol{h}_i \in S}$

$U, \Lambda, V \leftarrow \text{SVD}_k(S_1)$

$C_1, C_2 \leftarrow$ 2-means$(\{U^\top \boldsymbol{h} \mid \boldsymbol{h} \in S_1\})$

return clusters C_1, C_2

Chen et al. [48] propose several methods to determine which clusters, if any, contain poisoned representations. To avoid these complexities, we equip the algorithm with an oracle, ClusterOracle, which given two clusters returns the cluster with the greatest fraction of poisoned examples. The algorithm which returns the best cluster out of C_1, C_2 give by the oracle should perform at least as well as any heuristic to determine which clusters to return. There are two other concerns which make it difficult to compare this defense with Algorithm 1: first, there is no way to control how many examples are removed and second, the performance of the clustering varies with the initialization of k-means, which is random. Therefore, we use a second step which repeatedly runs Algorithm 10 and samples the cluster with the highest fraction of poison according to the oracle in order to build the set of samples to remove. The algorithm is shown in Algorithm 11.

Algorithm 11 should perform well whenever the clustering is able to effectively separate the poisoned examples from clean ones and its performance should have relatively low variance as R is built using many independent clustering runs. Although this process is not guaranteed to terminate, we found that it did in all of our experiments.

Algorithm 11: Activation Clustering with Cluster Oracle

Input: representation $S = \{\boldsymbol{h}_i \in \mathbb{R}^d\}_{i=1}^n$, dimension k

$R \leftarrow \varnothing$

while $|R| < 1.5\varepsilon n$ **do**

  $C_1, C_2 \leftarrow \textsc{ActivationClustering}(S, k)$ [Algorithm 10]

  $C \leftarrow \textsc{ClusterOracle}(C_1, C_2)$

  Sample $\boldsymbol{h}$ uniformly from C

  Add $\boldsymbol{h}$ to R if $\boldsymbol{h} \notin R$

return samples corresponding to R

A.2 Robust Estimation

We reproduce details from [67] which are relevant to the implementation and usage of Algorithm 1 here for completeness. First, we introduce some notations. Given two sets A and B, $\Delta(A, B)$ is the size of their symmetric difference $|(A \setminus B) \cup (B \setminus A)|$. Given a matrix $M \in \mathbb{R}^{d \times d}$, we write $M^\flat$ to denote the flattened vector $\boldsymbol{v} \in \mathbb{R}^{d^2}$ built by concatenating the columns of M. Similarly, given a vector $\boldsymbol{v} \in \mathbb{R}^{d^2}$, we write $\boldsymbol{v}^\sharp$ to denote the matrix $M \in \mathbb{R}^{d \times d}$ with $\boldsymbol{v}_i$ as columns, where $\boldsymbol{v}$ is split into d contiguous vectors in $\mathbb{R}^d$.

A.2.1 Robust Mean Estimation

There exists a practical robust mean estimation algorithm RobustMean which is given explicitly in Algorithm 12.

Understanding Algorithm 12 requires the definition of an (ε, τ)-good set with respect to a Gaussian, which is given in Definition A.1. The key feature of (ε, τ)-goodness is that a set of independent samples from the Gaussian of sufficient size is (ε, τ)-good with high probability as stated in Lemma A.2.

Definition A.1. [67, Definition A.4] Let G be a sub-gaussian distribution in d dimensions with mean $\boldsymbol{\mu}(G)$ and covariance matrix I and let $\varepsilon, \tau > 0$. We say that a multiset S of

Algorithm 12: Robust mean estimation (RobustMean) [67]

Input: A multiset S' such that there exists an (ε, τ)-good set S with $\Delta(S, S') < 2\varepsilon$

Output: A vector $\boldsymbol{\mu}'$ such that $\|\boldsymbol{\mu}' - \boldsymbol{\mu}(G)\|_2 \leq O(\varepsilon\sqrt{\log(1/\varepsilon)})$

repeat

  $S' \leftarrow$ GaussianMeanFilter(S') [Algorithm 13]

until *GaussianMeanFilter returns* $\boldsymbol{\mu}'$

return $\boldsymbol{\mu}'$

elements in $\mathbb{R}^d$ is (ε, τ)-good with respect to G if the following conditions are satisfied:

1. For all $\boldsymbol{x} \in S$ we have $\|\boldsymbol{x} - \boldsymbol{\mu}(G)\|_2 \leq O(\sqrt{d\log(|S|/\tau)})$.

2. For every affine function $L : \mathbb{R}^d \to \mathbb{R}$ such that $L(\boldsymbol{x}) = \boldsymbol{v} \cdot (\boldsymbol{x} - \boldsymbol{\mu}(G)) - T$, $\|\boldsymbol{v}\|_2 = 1$, we have that

$$|\mathsf{P}_{X \in_u S}[L(X) \geq 0] - \mathsf{P}_{X \sim G}[L(X) \geq 0]| \leq \frac{\varepsilon}{T^2 \log\big(d\log(\frac{d}{\varepsilon\tau})\big)}$$

3. We have that $\|\boldsymbol{\mu}(S) - \boldsymbol{\mu}(G)\|_2 \leq \varepsilon$.

4. We have that $\|M_s - I\|_2 \leq \varepsilon$.

Lemma A.2. *[67, Lemma A.6] Let G be a sub-gaussian distribution with parameter $\nu = \Theta(1)$ and identity covariance and let $\varepsilon, \tau > 0$. If the multiset S is obtained by taking $\Omega((d/\varepsilon^2)\operatorname{poly}\log(d/\varepsilon\tau))$ independent samples from G, it is ε-good with respect to G with probability at least $1 - \tau$.*

Now we give the definition of the filter used in Algorithm 12 in Algorithm 13, which shows that the sets S' in Algorithm 12 approach the ε-good set S with respect to the size of their symmetric difference.

Algorithm 13: Filter algorithm for a Gaussian with unknown mean. [67, Algorithm 2]

Input: A multiset S' such that there exists an (ε, τ)-good set S with $\Delta(S, S') < 2\varepsilon$

Output: Either a set S'' with $\Delta(S, S'') \leq \Delta(S, S') - \varepsilon/\alpha$ where

$\alpha \triangleq d \log(d/\varepsilon\tau) \log(d \log(d/\varepsilon\tau))$ or a vector $\boldsymbol{\mu}$ satisfying

$\|\boldsymbol{\mu}' - \boldsymbol{\mu}(G)\|_2 \leq O(\varepsilon\sqrt{\log(1/\varepsilon)})$

Compute the sample mean $\boldsymbol{\mu}(S') = \mathbb{E}_{X \sim \mathrm{Unif}(S')}[X]$.

Compute the sample covariance matrix $\Sigma(S') = \mathbb{E}_{X \in \mathrm{Unif}(S')}[(X - \boldsymbol{\mu}(S'))(X - \mu(S'))^\top]$.

Compute an approximation of the largest absolute eigenvalue of $\Sigma - I$, $\lambda^* \approx \|\Sigma - I\|_2$ and an approximate associated eigenvector $\boldsymbol{v}^*$.

if $\lambda^* \leq O(\varepsilon \log(1/\varepsilon))$ **then**

  return $\boldsymbol{\mu}(S')$

Let $\delta = 3\sqrt{\varepsilon\lambda^*}$. Find a $T > 0$ such that

$$\mathsf{P}_{X \in \mathrm{Unif}(S')}(|\boldsymbol{v}^* \cdot (X - \boldsymbol{\mu}(S'))| > T + \delta) > 8 \exp\left(-\frac{T^2}{2\nu}\right) + \frac{8\varepsilon}{T^2 \log(d \log(\frac{d}{\varepsilon\tau}))}.$$

return $S'' = \{\boldsymbol{x} \in S'' : |\boldsymbol{v}^* \cdot (\boldsymbol{x} - \boldsymbol{\mu}(S'))| \leq T + \delta\}$

A.2.2 Robust Covariance Estimation

The structure of this subsection mirrors that of Section A.2.1. Theorem 2.1 states the existence of a practical robust covariance estimation algorithm RobustCov which is given explicitly in Algorithm 14.

Understanding Algorithm 14 requires the definition of an ε-good set with respect to a Gaussian, which is given in Definition A.3. The key feature of ε-goodness is that a set of independent samples from the Gaussian of sufficient size is ε-good with high probability as stated in Proposition A.4.

Definition A.3. [67, Definition A.27] Let G be a Gaussian in $\mathbb{R}^d$ with mean 0 and covariance

Algorithm 14: Robust covariance estimation (RobustCov) [67]

Input: A multiset S' such that there exists an ε-good set S with $\Delta(S, S') < 2\varepsilon$

Output: A matrix Σ' such that $\|I - \Sigma^{-1/2}\Sigma'\Sigma^{-1/2}\|_F = O(\varepsilon \log(\frac{1}{\varepsilon}))$

repeat

  $S' \leftarrow$ GaussianCovarianceFilter(S') [Algorithm 15]

until *GaussianCovarianceFilter returns* Σ'

return Σ'

Σ. Let $\varepsilon > 0$ be sufficiently small. We say that a multiset S of points in $\mathbb{R}^d$ is ε-good with respect to G if the following hold:

1. For all $\boldsymbol{x} \in S$, $\boldsymbol{x}^\top \Sigma^{-1} \boldsymbol{x} < d + O(\sqrt{d} \log(d/\varepsilon))$.

2. We have that $\|\Sigma^{-1/2} \operatorname{Cov}(S) \Sigma^{-1/2} - I\|_F = O(\varepsilon)$.

3. For all even degree-2 polynomials p, we have that $\operatorname{Var}(p(\boldsymbol{x})) = \operatorname{Var}(p(G))(1 + O(\varepsilon))$.

4. For p an even degree-2 polynomial with $\mathbb{E}[p(G)] = 0$ and $\operatorname{Var}(p(G)) = 1$, and for any $T > 10 \log(1/\varepsilon)$ we have that

$$\mathsf{P}(|p(x)| > T) \leq \frac{\varepsilon}{T^2 \log^2(T)}.$$

Proposition A.4. *[67, Proposition A.28] Let N be a sufficiently large constant multiple of $(d^2/\varepsilon^2) \log^5(d/\varepsilon)$. Then a set S of N independent samples from G is ε-good with respect to G with high probability.*

Now we give the definition of the filter used in Algorithm 14 in Algorithm 15, which shows that the sets S' in Algorithm 14 approach the (ε, τ)-good set S with respect to the size of their symmetric difference.

Note that a naive implementation of Algorithm 16 requires $\Omega(nd^2)$ space to store the $\boldsymbol{y}_i$ and $\Omega(d^4)$ space to store $T_{S'}$. Additionally, the matrix multiplication performed by OpenBLAS

to produce $T_{S'}$ requires $\Omega(nd^4)$ time. By representing the linear operator $T_{S'}$ implicitly, we can reduce these requirements substantially. First, the product $-I^\flat(I^{\flat\top}\boldsymbol{v})$ can be computed in $O(d^2)$ time and space. Next, if Y and Z are the matrices with columns $\boldsymbol{y}_i$ and $\boldsymbol{z}_i$ respectively, then Z is the Khatri-Rao product $Y \odot Y$. This means we can use the vec tricks for the Khatri-Rao and transpose Khatri-Rao vector products of [194] to calculate $ZZ^\top\boldsymbol{v}$ in $O(nd^2)$ time and $O(nd + d^2)$ space. We can then calculate the eigenvector $\boldsymbol{v}^*$ of the implicitly represented linear operator $T_{S'}$ using Krylov methods, requiring the evaluation of a small number of products $T_{S'}\boldsymbol{v}$. For our experiments, this provided a speedup of several orders of magnitude and a substantial reduction in the required amount of system memory versus the naive implementation.

Algorithm 15: Filter algorithm for a Gaussian with unknown covariance matrix. [67, Algorithm 4]

Input: A multiset S' such that there exists an ε-good set S with $\Delta(S, S') < 2\varepsilon$

Output: Either a set S'' with $\Delta(S, S'') < \Delta(S, S')$ or a matrix Σ' such that

$$\|I - \Sigma^{-1/2}\Sigma'\Sigma^{-1/2}\|_F = O(\varepsilon \log(\tfrac{1}{\varepsilon}))$$

Let $C, C' > 0$ be sufficiently large universal constants.

$\Sigma' \leftarrow \mathbb{E}_{X \in S'}[XX^\top]$, $G' \leftarrow \mathcal{N}(0, \Sigma')$

if *there exists an* $\boldsymbol{x} \in S'$ *such that* $\boldsymbol{x}^\top \Sigma'^{-1} \boldsymbol{x} \geq Cd \log(10|S'|)$ **then**

  return $S'' = S' \setminus \{\boldsymbol{x} \in S' : \boldsymbol{x}^\top \Sigma'^{-1} \boldsymbol{x} > Cd \log(10|S'|)\}$

Let L be the space of even degree-2 polynomials $p : \mathbb{R}^k \to \mathbb{R}$ such that $\mathbb{E}_{X \sim G'}[p(X)] = 0$.

Define two quadratic forms on L:

(i) $Q_{G'}(p) = \mathbb{E}_{X \sim G'}[p^2(X)]$

(ii) $Q_{S'}(p) = \mathbb{E}_{X \sim \mathrm{Unif}(S')}[p^2(X)]$

Compute $\max_{p \in L \setminus \{0\}} Q_{S'}(p)/Q_{G'}(p)$ and the associated polynomial $p^*(x)$ normalized such that $Q_{G'}(p) = 1$ using Algorithm 16.

if $Q_{S'}(p^*) \leq (1 + C\varepsilon \log^2(1/\varepsilon)) Q_{G'}(p^*)$ **then**

  return Σ'

$\mu \leftarrow$ the median value of $p^*(X)$ over $X \in S'$

Find a $T > C'$ such that

$$\mathsf{P}_{X \in T'}(|p^*(X) - \mu| \geq 3) \leq \mathrm{Tail}(T, d, \varepsilon),$$

where

$$\mathrm{Tail}(T, d, \varepsilon) = \begin{cases} 3\varepsilon/(T^2 \log^2(T)) & \text{if } T \geq 10 \ln(1/\varepsilon) \\ 1 & \text{otherwise} \end{cases}.$$

return $S'' = \{\boldsymbol{x} \in S'' : |p^*(U'^\top \boldsymbol{x}) - \mu| \leq T\}$

Algorithm 16: Algorithm to compute the polynomial with maximum variance relative to a Gaussian [67, Algorithm 4]

Input: A multiset $S' = \{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^d$ and a Gaussian $G' = \mathcal{N}(0, \Sigma')$

Output: The even degree-2 polynomial $p^*(\boldsymbol{x})$ with $\mathbb{E}_{X \sim G'}[p(X)] \approx 0$ and $Q_{G'}(p^*) \approx 1$ that approximately maximizes $Q_{S'}(p^*)$ and this maximum is $\lambda^* = Q_{S'}(p^*)$

for $i \in [n]$ **do**

  $\boldsymbol{y}_i \leftarrow \Sigma_k'^{-1/2} \boldsymbol{x}_i$

  $\boldsymbol{z}_i \leftarrow (\boldsymbol{y}_i \boldsymbol{y}_i^\top)^\flat$

$T_{S'} \leftarrow -I^\flat I^{\flat\top} + \frac{1}{|S'|} \sum_{i=1}^n \boldsymbol{z}_i \boldsymbol{z}_i^\top$

Approximate the top eigenvalue λ^* and eigenvector $\boldsymbol{v}^*$ of $T_{S'}$

$p*(x) \leftarrow \frac{1}{\sqrt{2}}((\Sigma_k'^{-1/2}\boldsymbol{x})\boldsymbol{v}^{*\sharp}(\Sigma_k'^{-1/2}\boldsymbol{x}) - \mathrm{Tr}(\boldsymbol{v}^{*\sharp}))$

return p^* and λ^*

A.2.3 Robust Joint Mean and Covariance Estimation

Note that Algorithm 12 requires the inputs to have identity covariance and Algorithm 14 requires the inputs to have zero mean. Here we show how to combine them to estimate both the mean and covariance of an arbitrary Gaussian, as described in [67, Section 4.5]. The key idea is to split the dataset into two halves, pair off samples from each half, and subtract them. The resulting vectors have zero mean and double the original covariance. This allows us to use Algorithm 14 to whiten the samples, which then allows us to use Algorithm 12. We reproduce the exact procedure in Algorithm 17.

Algorithm 17: Algorithm to robustly learn an arbitrary Gaussian [69, Algorithm 6]

Input: A multiset $S' = \{\boldsymbol{x}_i\}_{i=1}^n \subset \mathbb{R}^d$, corruption fraction ε

Output: A matrix Σ' such that $\|I - \Sigma^{-1/2}\Sigma'\Sigma^{-1/2}\|_F = O(\varepsilon \log(\frac{1}{\varepsilon}))$ and Aavector $\boldsymbol{\mu}'$ such that $\|\boldsymbol{\mu}' - \boldsymbol{\mu}(G)\|_2 \leq O(\varepsilon\sqrt{\log(1/\varepsilon)})$

for $i \in [\lfloor n/2 \rfloor]$ **do**

  $\boldsymbol{x}'_i \leftarrow (\boldsymbol{x}_i - \boldsymbol{x}_{\lfloor n/2 \rfloor + 1})/\sqrt{2}$

$\widehat{\Sigma} \leftarrow \textsc{RobustCov}(\{\boldsymbol{x}'_i\}, \varepsilon)$ [Algorithm 14]

for $i \in [n]$ **do**

  $\boldsymbol{x}''_i \leftarrow \widehat{\Sigma}^{-1/2}\boldsymbol{x}_i$

$\widehat{\boldsymbol{\mu}} \leftarrow \textsc{RobustMean}(\{\boldsymbol{x}''_i\}, \varepsilon)$ [Algorithm 12]

return $\widehat{\Sigma}$ and $\widehat{\Sigma}^{1/2}\widehat{\boldsymbol{\mu}}$

A.3 Experiment Details

For each poisoned dataset, we performed one training run to produce each poisoned model. For the pixel and periodic attacks, we performed one retraining run for each defense. Training for our experiments was done on a server with a Xeon Gold 6230 CPU and eight Nvidia 2080 Ti GPUs. The training and retraining for our experiments took approximately 100 GPU hours. Running all defences for our experiments took approximately 200 CPU-core hours.

Using the thermal design power of these components to estimate of our required power, we estimate that our experiments required a total of 28 kW h of energy.

A.3.1 m-Way Pixel Attacks

For pixel attacks, we reproduce the experimental setup of [241]. For our ResNet-32, we used a leaky ReLU with a negative slope of 0.1 for the nonlinearity and trained it using stochastic gradient descent with momentum for 200 epochs, dividing the learning rate by 10 every 75 epochs. Both data standardization and augmentation were used.

Although a fixed pixel is used for watermarking, data augmentation may ensure that the network is sensitive to pixels of the chosen color at multiple locations in the image. Using the standard random horizontal flip and random crop with 4 pixels of padding used for CIFAR-10, the pixel may end up in as many as $9 \times 9 \times 2 = 162$ distinct pixels in the transformed image, representing about 16% of the image's total area.

To implement an m-way pixel attack, m pairs of locations and colors are chosen. Only one of the m pixels is used for each poisoned training example, but all m are used simultaneously at test time. We ran experiments for $m \in \{1, 2, 3\}$. We used the same backdoor pixel Tran et al. used for their experiments, along with two more arbitrarily chosen. The exact locations and colors are shown in Table A.1.

m	location	color
1	(11, 16)	#650019
2	(5, 27)	#657B79
3	(30, 7)	#002436

Table A.1: **Pixel watermarks used for the m-way pixel attacks**: Location is a pixel coordinate in (x, y) format and color is a 24-bit hexadecimal color in HTML format.

A.3.2 m-Way Periodic Attacks

For periodic attacks, we used the same network architecture and training environment used for pixel attacks. Although the phase of the signal is fixed for watermarking, the signal will be shifted by a random amount at training time due to the random flip and random crop and pad, in a manner similar to the pixel attack. Because our signals have a period of 4 pixels, which equals the maximum translation produced by the data augmentation, the backdoored network should be sensitive to signals with any phase.

A.3.3 Label Consistent Attacks

For label consistent attacks we used the experimental setup of [246] which is provided at `https://github.com/MadryLab/label-consistent-backdoor-code`. The setup of [246] appears to be very similar to that of [241]. The same ResNet-32 architecture is used, albeit with a normal (i.e. not leaky) ReLU. Data standardization was enabled by default. Data augmentation was disabled by default, but we enabled it to ensure greater consistency with our previous experiments. We also enabled patch placement on all four corners to ensure the watermark would not be cropped out. For this family of attacks, we did not make any changes to the training system of [246], which does not provide retraining.

A.4 Analysis of Poisoned Representations

Here we include Figs. A.1 to A.4, which illustrate some relevant properties of the hidden layer activations of examples bearing the target layer under a successful backdoor poisoning attack.

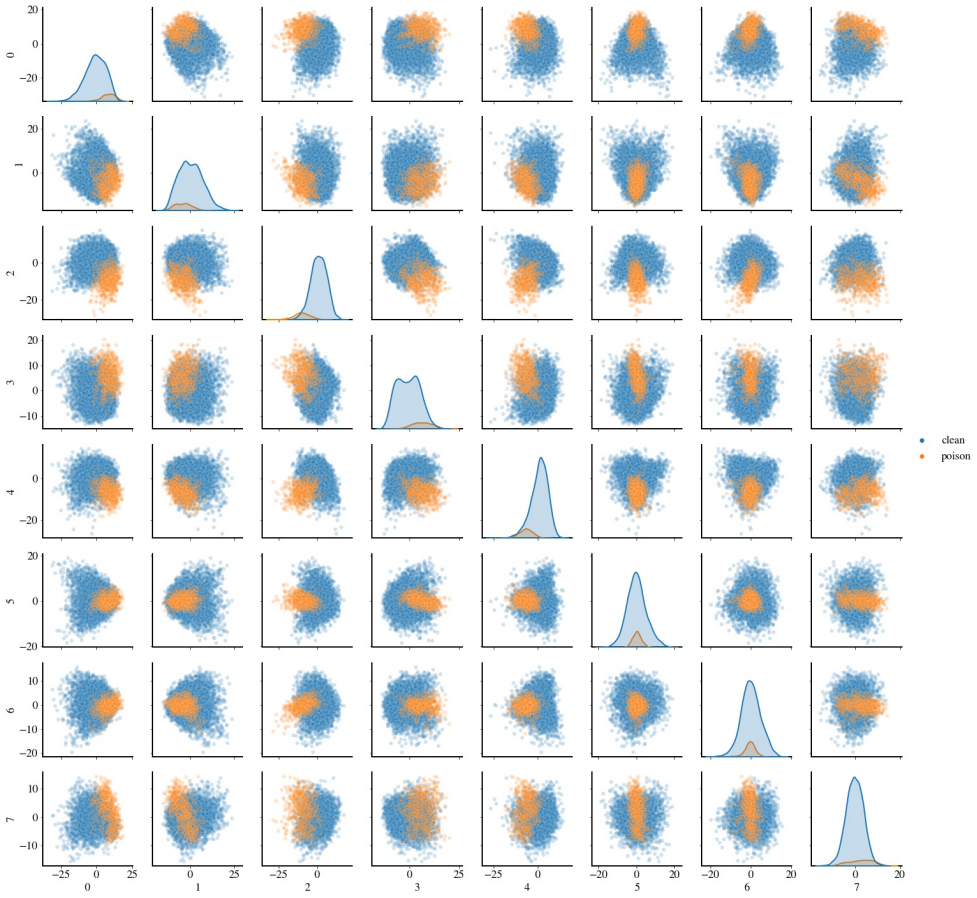


Figure A.1: **Scatter plots of the representations of the 3-way pixel attack with $\varepsilon = 0.1$ before any whitening**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction (of the combined data including the representations of both the poisoned and the clean samples). Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the corresponding pair of PCA directions. This shows that the poisoned samples (in orange) are not separable from the clean ones (in blue), if we only focus on these top PCA directions; the spectral signature is hidden. We propose using robust covariance estimation to fine the approximate covariance of clean data and whiten the entire data with the estimated covariance. This enhances the spectral signature as we show in the next figure.

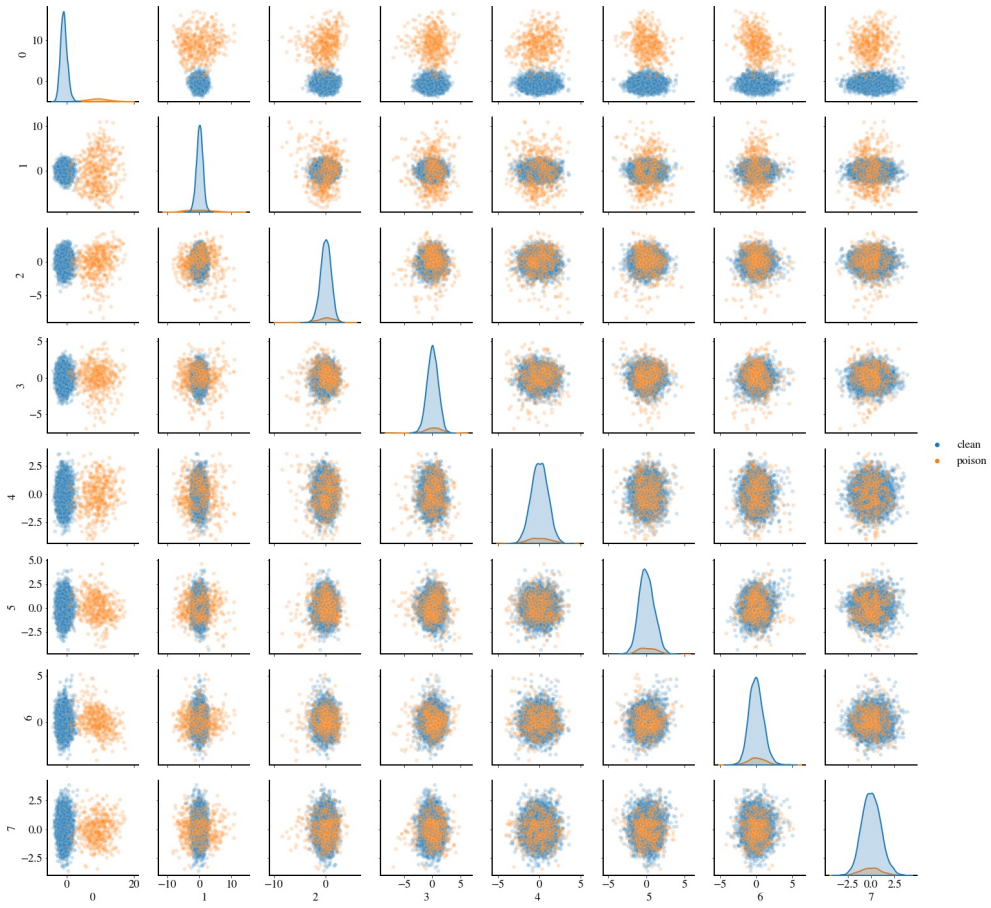


Figure A.2: **Scatter plots of the representations of the 3-way pixel attack with $\varepsilon = 0.1$ after whitening using the covariance of the clean samples**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction (of the combined data including the representations of both the poisoned and the clean samples). Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the corresponding pair of PCA directions. This shows that the poisoned samples (in orange) are now separable from the clean ones (in blue) using the top PCA direction after whitening, for example.

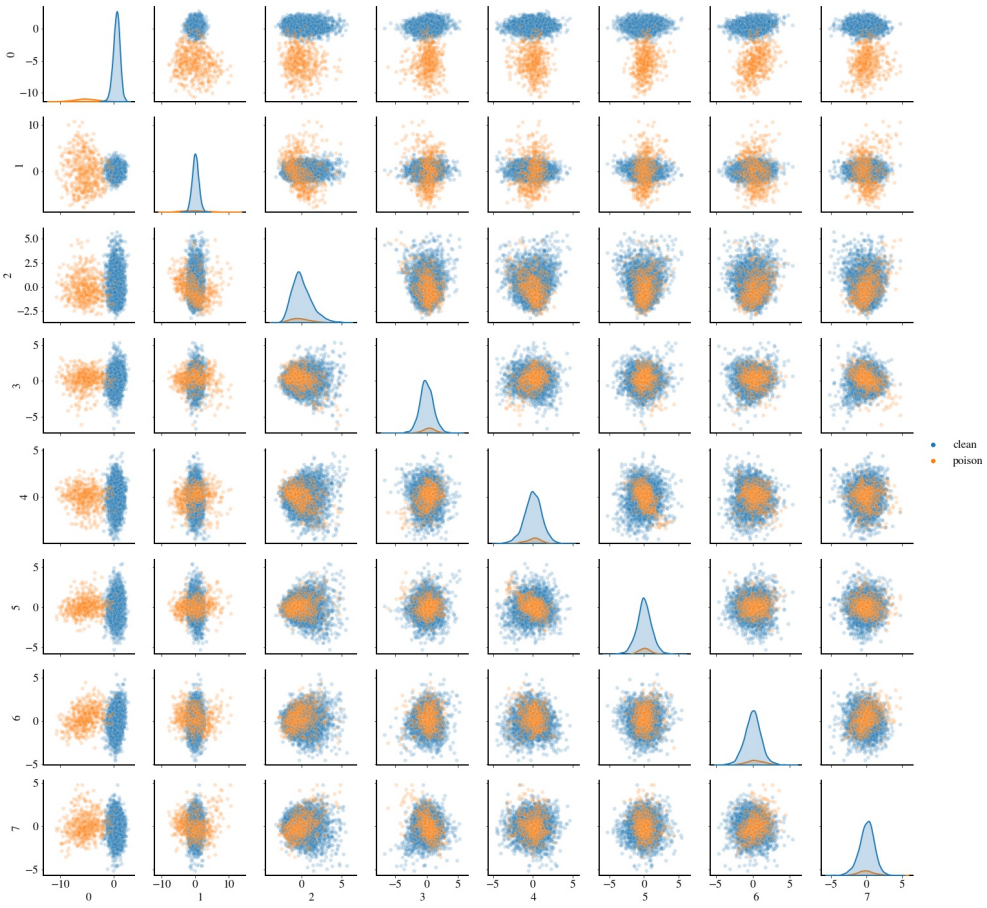


Figure A.3: **Scatter plots of the representations of the 3-way pixel attack with $\varepsilon = 0.1$ after whitening using the robustly estimated covariance**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction (of the combined data including the representations of both the poisoned and the clean samples). Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the corresponding pair of PCA directions. This shows that the poisoned samples (in orange) remain separable from the clean ones (in blue) even when whitening using the estimated covariance instead of the true covariance of the clean samples.

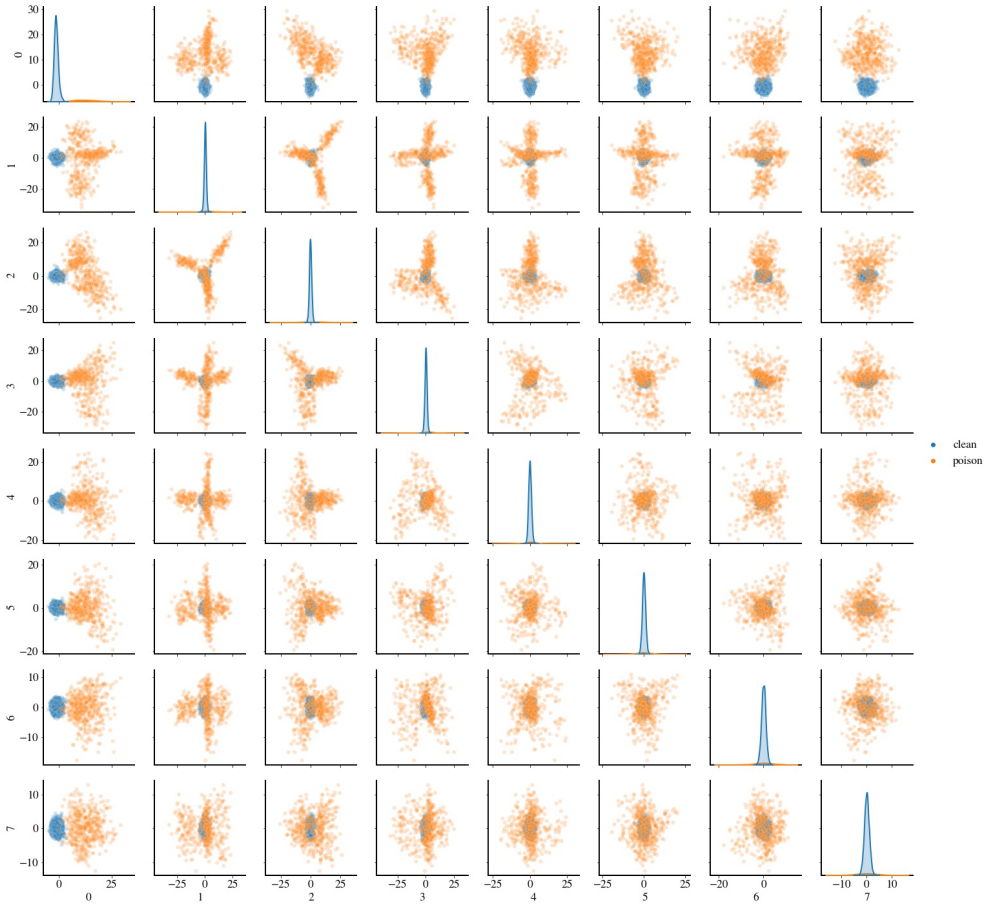


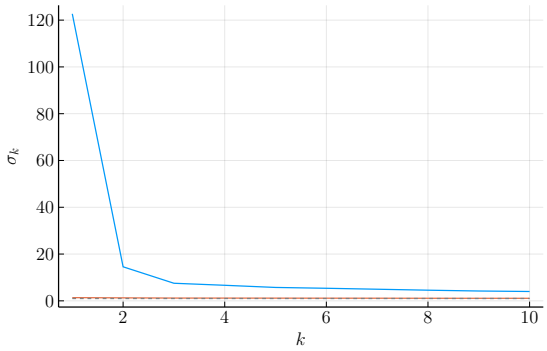
Figure A.4: **Scatter plots of the representations of the 2-way pixel attack with $\varepsilon = 0.1$ after whitening with the true covariance of the representation of the clean samples**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction. Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the corresponding pair of PCA directions. This shows that the poisoned samples (in orange) have split into multiple distinct clusters, resulting in a weakened spectral signature. Nevertheless, whitening enhances the spectral signature and bring the direction of separation to the top principal components.

A.5 Analysis of QUE Scores

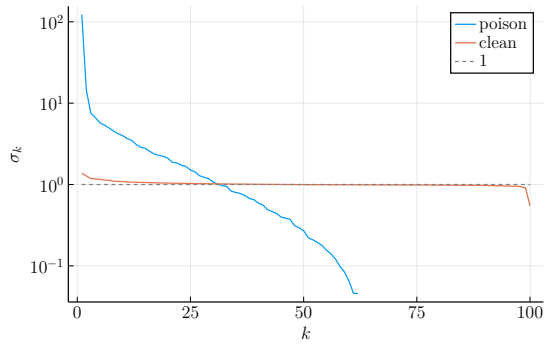
In Section 2.3.3, we showed that the squared norm scoring $\tau_i^{(0)} = \|\tilde{\boldsymbol{h}}_i\|^2$ and squared projected norm scoring $\tau_i^{(\infty)} = |\langle \boldsymbol{v}, \tilde{\boldsymbol{h}}_i \rangle|^2$ can both fail under certain conditions. Here we will explain those conditions in greater detail.

In our experiments, squared norm scoring $\tau_i^{(0)} = \|\tilde{\boldsymbol{h}}_i\|^2$ fails for the 3-way attack with $\varepsilon = 0.0124$. For this attack, the poisoned representations have high variance along a single direction, and relatively low variance along all other directions, as seen in Fig. A.5a. Because there are few poisoned examples relative to clean ones, the resulting spectral signature of the poisoned examples is weak. The directions where the variance of the clean data was amplified, as seen in Fig. A.5b, dominate all but one of the directions where the poison had high variance. This can be seen in Fig. A.6, where only the top PCA direction, which corresponds to projected norm scoring, is suitable for removing the poisoned examples. Using the squared norm scoring $\tau_i^{(0)} = \|\tilde{\boldsymbol{h}}_i\|^2$ here causes the top PCA direction to be mixed with the less useful directions, diluting its utility as a metric for removing the poison.

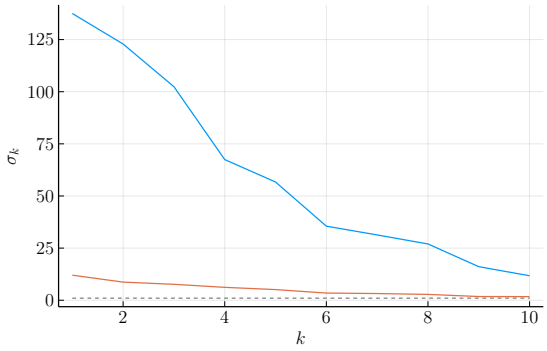
Squared projected norm scoring $\tau_i^{(\infty)} = |\langle \boldsymbol{v}, \tilde{\boldsymbol{h}}_i \rangle|^2$ fails for the 1-way attack with $\varepsilon = 0.1$. Here the spectral signature of the poisoned examples is very strong. The poisoned examples have high variance along many directions, as seen in Fig. A.5c. The resulting top PCA direction $\boldsymbol{v}$ is not well aligned with the direction of the separation $\boldsymbol{\mu}(S_{\text{poison}}) - \boldsymbol{\mu}(S_{\text{clean}})$. In fact, the angle between them is $\cos^{-1}(\langle \boldsymbol{v}, \boldsymbol{\mu}(S_{\text{poison}}) - \boldsymbol{\mu}(S_{\text{clean}}) \rangle / \|\boldsymbol{\mu}(S_{\text{poison}}) - \boldsymbol{\mu}(S_{\text{clean}})\|) = 35.7°$. The consequence of this misalignment can be seen in Fig. A.7, where it is clear that $\boldsymbol{v}$ does not separate the poisoned examples from the clean ones. On the other hand, squared norm scoring works well here because the poisoned examples have large variance along many directions, which is apparent in Fig. A.7.



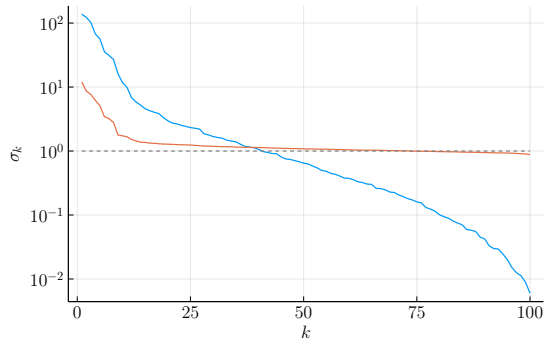
(a) 3-way attack with $\varepsilon = 0.0124$, top 10 singular values



(b) 3-way attack with $\varepsilon = 0.0124$, top 100 singular values



(c) 1-way attack with $\varepsilon = 0.1$, top 10 singular values



(d) 1-way attack with $\varepsilon = 0.1$, top 100 singular values

Figure A.5: **Plots of the top 10/100 singular values of the covariances of the poison and clean representations after whitening with the robustly estimated covariance in order of decreasing magnitude.**

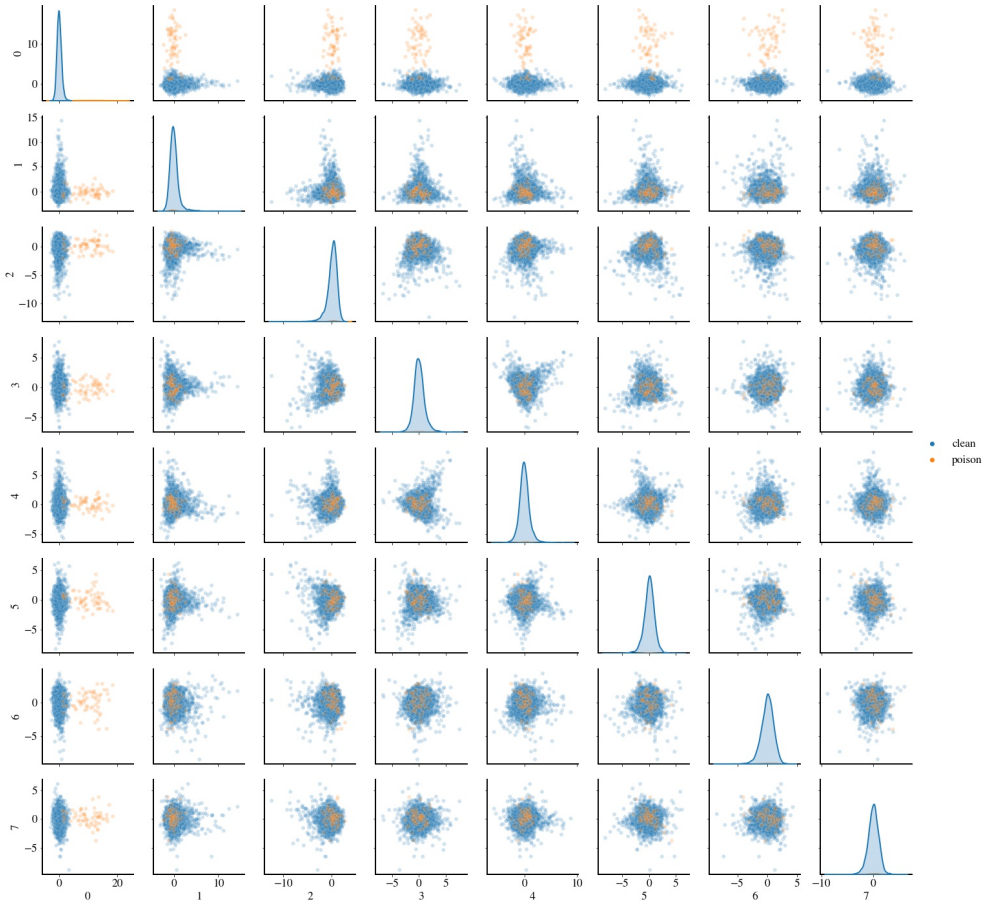


Figure A.6: **Scatter plots of the representations of the 3-way pixel attack with $\varepsilon = 0.0124$ after robust whitening; whitening the representations of the data with the estimated covariance of the clean samples**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction. Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the corresponding pair of PCA directions. This clearly shows that the top PCA direction is aligned with the direction of separation between the poisoned samples (in orange) and clean samples (in blue), hence the squared projected norm scoring works. However, the variance of the poisoned examples are generally smaller, making it hard to distinguish using the squared norm scoring.

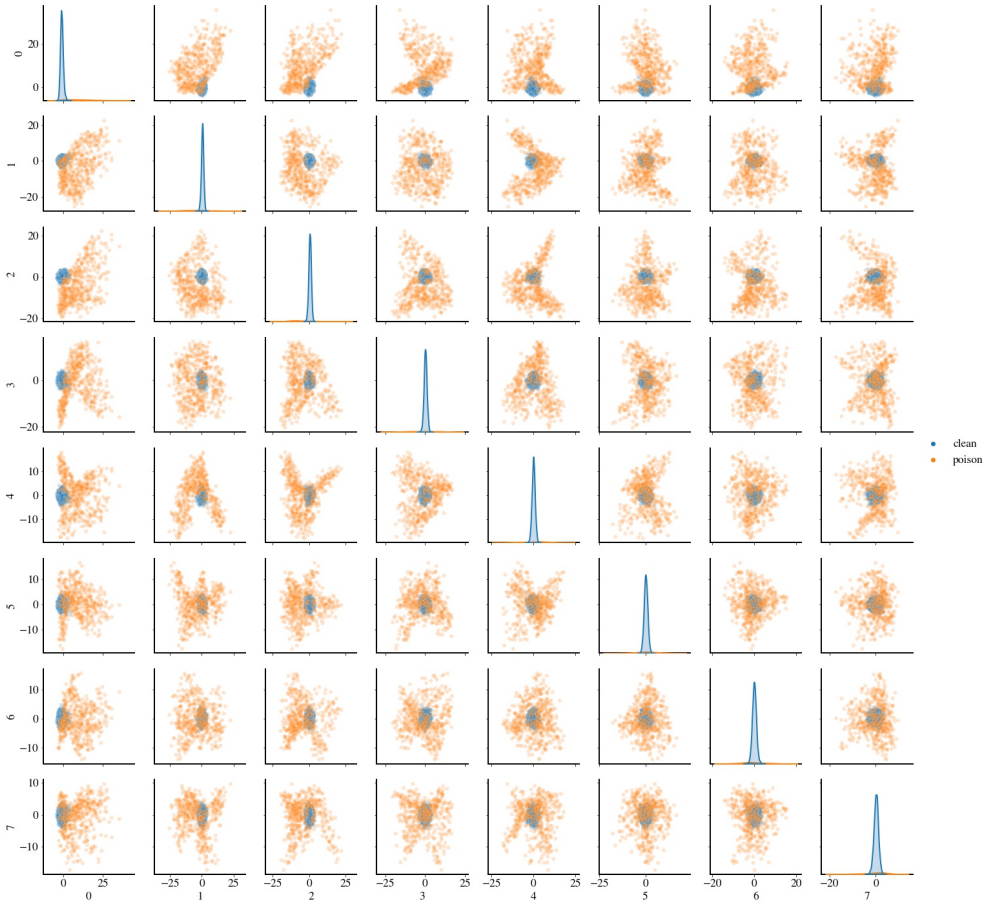


Figure A.7: **Scatter plots of the representations of the 1-way pixel attack with $\varepsilon = 0.1$ after robust whitening; whitening the representations of the data with the estimated covariance of the clean samples**: The whitened representations are projected onto their top eight PCA directions. Plots along the diagonal are Gaussian kernel density estimate plots after projecting onto that PCA direction. Off-diagonal plots are scatter plots of the data projected onto the subspace spanned by the pair of PCA directions. This clearly shows that the top PCA direction is not aligned with the direction of separation between the poisoned samples (in orange) and clean samples (in blue), hence the squared projected norm scoring does not works. However, the variance of the poisoned examples are generally larger, making it easy to distinguish using the squared norm scoring.

A.6 Sensitivity to Number of Removed Examples

Following [241], we choose to remove the $1.5\varepsilon n$ samples with the highest QUE scores from the $(1+\varepsilon)n$ total samples bearing the target label. We show in Fig. A.8 that our defence performance is not overly sensitive to this choice. In particular, the fraction of poisoned samples removed does not vary substantially with the total number of removed samples after the first εn samples are removed.

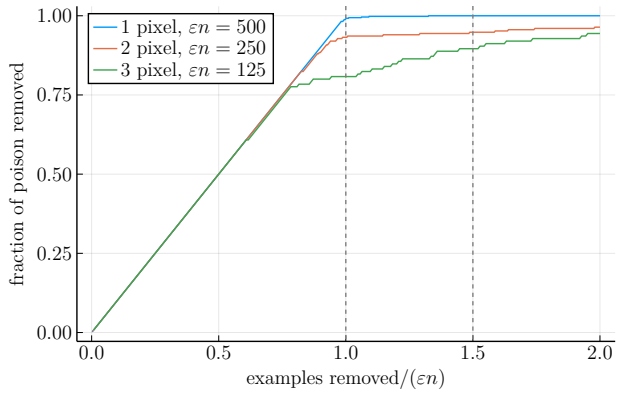


Figure A.8: **Fraction of all poisoned samples removed vs. the total number of samples removed by SPECTRE, for three pixel attacks featuring spectral signatures of varying strength.**

Appendix B

FEW-SHOT BACKDOOR ATTACKS VIA NEURAL TANGENT KERNELS

B.1 Attack Model and Related Work

We survey relevant train-time attacks.

B.1.1 Threat Model

We consider the following train time attack known as the backdoor attack. We assume that the attacker has a trigger function of choice and their goal is to corrupt the training of a model such that the corrupted model outputs a prediction of a target label when shown an example with the trigger function applied to it. The attacker can inject a small number of arbitrarily corrupted examples to the training set to achieve this goal. The success will be measures by Attack Success Rate (ASR), which is the probability of the prediction of a trigger-applied test example being the target label. One attack, which consists of a set of a fixed number of injected corrupted examples, is said to be stronger than another if the ASR is higher than the other given the same number of injected corrupted examples.

We assume that the attacker has knowledge of the target model's architecture and training data, and can and leverage this information to increase the ASR of the backdoor attacks. In Section B.1.2 we list several prior works make similar assumptions for both data-poisoning and backdoor attacks. We believe knowledge of the training architecture can be motivated by the widespread usage of popular architectures such as ResNets [104]. We also perform an ablation study in Section 3.3.3 indicating that our method degrades gracefully when only a small subset of the training data is known.

B.1.2 Backdoor Attacks

Backdoor attacks as presented in Section 3.1 are introduced in [95]. In backdoor attacks, the two most important design choices are the choice of trigger P and the method of producing the poison data X_{p}. Many works design P to appear benign to humans [95, 26, 157, 184] or directly optimize P to this end [140, 71]. Poison data X_{p} has been constructed to include no mislabeled examples [246, 281] and optimized to evade detection through visual inspection [209] and statistical inspection of latent representations [221, 70, 258, 53]. Such backdoor attacks have been demonstrated in a wide variety of settings, including federated learning [254, 21, 224], transfer learning [271, 209], and generative models [210, 203]. However, our goal of designing strong *few-shot backdoor attacks* has not been addressed with an exception of an influential earlier work of [122]. We consider the same threat model as in [122] where the attacker has information about the network's architecture and training data. However, our results are incomparable to those of [122] which focuses on linear models. The KKT attack of [122] leveraging decoy parameters cannot be used when the input dimension is far smaller than the parameter dimension and the influence attack of [122] cannot scale to large models, such as the WideResNet we use in our experiments.

Few-shot data attacks have been studied in contexts other than backdoor attacks. In *targeted* backdoor attacks, the attacker aims to control the network's output on a specific test instance [216, 26, 98, 5, 108]. *Data poisoning attacks* are similar to backdoor attacks with the alternate goal of reducing the generalization performance of the resulting model. Poison data X_{p} has been optimized to produce stronger data poisoning attacks using influence functions [122, 268], back-gradients [178], and the neural tangent kernel [275].

Following [95], there has also been substantial work on detecting and defending against backdoor attacks. When the defender has access to known-clean data, they can filter the data using outlier detection [145, 135, 223], retrain the network so it forgets the backdoor [153], or train a new model to test the original for a backdoor [123]. Other defenses assume P is an additive perturbation with small norm [253, 61], rely on smoothing [252, 256], filter or

penalize outliers without clean data [86, 224, 223, 37, 197, 241, 101] or use Byzantine-tolerant distributed learning techniques [37, 11, 50]. Backdoors cannot be detected in planted neural networks in general [93].

B.2 Implementation Details

In Section 3.2, we give a brief description of the Neural Tangent Backdoor Attack. Further details regarding the implementation are given here.

B.2.1 Optimization Details

We use L-BFGS-B by adapting the wrapper of [250] for use with JAX. We found that simple first order methods such as gradient descent with momentum and Adam [121] converged very slowly with small learning rates and were unable to achieve good minima with larger learning rates. In contrast, the strong Wolfe line search of L-BFGS-B appears to choose step sizes which lead to relatively rapid convergence for our problem.

B.2.2 Computational Resources

All neural networks were trained on a single Nvidia 2080 Ti. We ran NTBA optimization on a machine with four Nvidia A100 GPUs for a duration between 5 hours and 12 hours depending on the number of poisons being optimized. Before optimization begins, we precompute the $K_{\text{d,dta}}$ matrix using Nvidia A100 GPUs, requiring a total of 2 GPU hours for double precision.

B.2.3 Details for NTK and Laplace Kernel Comparison

In these experiments, we compare the NTK of a 3-layer feed-forward neural network with a Laplace kernel with bandwidth tuned to match the NTK in the small-distance limit. This is the same setup used in Section B.5. For the attack, we use the same parameters as for our main experiments in Table 3.2, except we do not report transfer numbers since the Laplace kernel has no associated neural network.

B.2.4 Details for Label Consistent Attack

In Fig. 3.1, we compare the NTBA to the sampling baseline as well as the label consistent attack of [246]. To produce the label consistent numbers, we use the precomputed poisoned datasets provided in the paper for the ℓ_∞ adversarial perturbations with $\varepsilon = 8$ and GAN latent interpolation with $\tau = 0.3$. The line reported is the best poison test accuracy obtained over these two attacks.

B.3 Supplementary Experimental Results

We report further experimental results complimenting those of Section 3.3.

B.3.1 Results for Patch Trigger on CIFAR-10

We repeat the experiments of Section 3.3.1 using a 3×3 checkered patch as the backdoor trigger. Example images for this attack are shown in Fig. B.1. We plot the ASR vs. the number of poisoned images in Fig. B.2 with numerical results reported in Table B.1.

(a) $m = 3$ (b) $m = 10$

Figure B.1: **NTBA poison images for patch triggers**: Images produced by backdoor optimization for the patch trigger and $m \in \{3, 10\}$. The top row shows the original clean image, the middle row shows the image with the trigger applied, and the bottom row shows the poisoned image after optimization. Duplicate images have been omitted to save space.

We note that for some images in Fig. B.1, the trigger becomes partially faded out after optimization while for other images the trigger remains unchanged. We believe this may be due to the optimization getting stuck in a local minima nearby some images, preventing it from erasing the triggers as we would expect according to the analysis in Section 3.4. This may partly explain why the attacks computed for the patch trigger are not as strong as those computed for the periodic trigger.

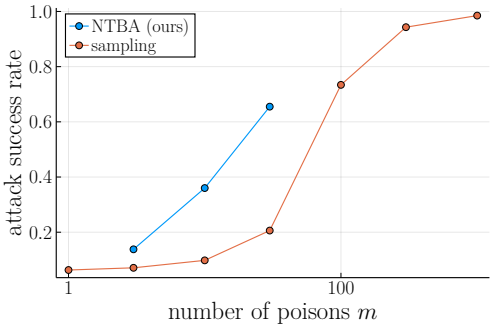


Figure B.2: **The trade-off between the number of poisons and ASR for the patch trigger.**

B.3.2 Results for Periodic Trigger on ImageNet

We also use NTBA to attack a ConvNeXt-tiny [158] ($d \approx 2.8 \times 10^7$) trained on a 2 label subset of ImageNet. We use "slot" as the source label and "Australian terrier" as the target label following the examples from [209]. We consider both the case where the ConvNeXt is initialized randomly and trained from scratch and the case where it has been pretrained on ImageNet and fine-tuned as in [209]. The results for these two settings are shown in Figs. B.3 and B.4 respectively. When trained from scratch, the clean accuracy of the ConvNeXt remains above 90% in all cases. When pretrained and fine-tuned, the ConvNeXt achieves at least 99% clean accuracy in all cases.

We note that ConvNeXt is surprisingly vulnerable to backdoors when trained from scratch,

	ours				sampling		sampling
m	$\text{asr}_{\text{ntk,tr}}$	$\text{asr}_{\text{ntk,te}}$	$\text{asr}_{\text{nn,tr}}$	$\text{asr}_{\text{nn,te}}$	$\text{asr}_{\text{nn,te}}$	m	$\text{asr}_{\text{nn,te}}$
3	99.9	74.1	0.9	13.8	7.1	0	6.2
10	99.0	79.8	37.7	36.0	9.8	1	6.3
30	93.8	82.3	66.8	65.5	20.6	100	73.4
						300	94.3
						1000	98.5

Table B.1: **NTBA attack success rates for patch triggers**: ASR of NTBA ($\text{asr}_{\text{nn,te}}$) is significantly higher than the ASR for the baseline of the sampling based attack using the same patch trigger, across a range of poison budgets m. Clean accuracy $\text{acc}_{\text{nn,te}}$ remains above 92.6% in all cases.

as even a single random poisoned image is sufficient to achieve 50% ASR and NTBA is able to achieve 100% ASR with a single image. With pretraining, the ConvNeXt becomes slightly more resistant to backdoors, but the periodic attack remains quite strong. We give numerical results in Table B.2.

B.4 Analysis of Backdoors for Kernel Linear Regression

To explain the phenomena observed in Section 3.4, we take Taylor approximations of ϕ at $\widetilde{\boldsymbol{x}}_{\text{p}}$ and $\widetilde{\boldsymbol{x}}_{\text{a}}$ and obtain,

$$
\begin{aligned}
&f(\boldsymbol{x}_{\text{a}}; D_{\mathsf{d}} \cup \{(\boldsymbol{x}_{\text{p}}, y_{\text{p}})\}) - f(\boldsymbol{x}_{\text{a}}; D_{\mathsf{d}}) \\
&\quad \approx \frac{(\phi(\widetilde{\boldsymbol{x}}_{\text{p}}) + \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{p}})\boldsymbol{\Delta}_{\text{p}})(I-P)(\phi(\widetilde{\boldsymbol{x}}_{\text{a}}) + \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{a}})\boldsymbol{\Delta}_{\text{a}})^{\top}}{(\phi(\widetilde{\boldsymbol{x}}_{\text{p}}) + \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{p}})\boldsymbol{\Delta}_{\text{p}})(I-P)(\phi(\widetilde{\boldsymbol{x}}_{\text{p}}) + \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{p}})\boldsymbol{\Delta}_{\text{p}})^{\top}} (y_{\text{p}} - f(\boldsymbol{x}_{\text{p}}; D_{\mathsf{d}})) \\
&\quad = \underbrace{\frac{\langle \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{p}})\boldsymbol{\Delta}_{\text{p}}, \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{a}})\boldsymbol{\Delta}_{\text{a}} \rangle_{(I-P)}}{\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\text{p}})\boldsymbol{\Delta}_{\text{p}}\|^2_{(I-P)}}}_{\triangleq A} \underbrace{(y_{\text{p}} - f(\boldsymbol{x}_{\text{p}}; D_{\mathsf{d}}))}_{\approx 2} ,
\end{aligned}
$$

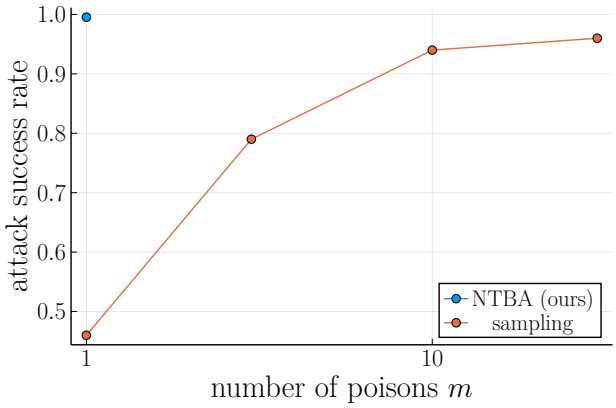


Figure B.3: **The trade-off between the number of poisons and ASR for ConvNeXt trained from scratch.**

where $\mathrm{D}\phi(\widetilde{\boldsymbol{x}})$ denotes the Jacobian of the feature mapping ϕ at $\widetilde{\boldsymbol{x}}$ and w.l.o.g. we assume that $y_\mathrm{p} = 1$ and $f(\boldsymbol{x}_\mathrm{p}; D_\mathsf{d}) \approx -1$. The last step follows because $(I - P)\phi(\widetilde{\boldsymbol{x}}_\mathrm{a}) = (I - P)\phi(\widetilde{\boldsymbol{x}}_\mathrm{p}) = \mathbf{0}$. Note that if $A = 1$, then $f(\boldsymbol{x}_\mathrm{a}; D_\mathsf{d} \cup \{(\boldsymbol{x}_\mathrm{p}, y_\mathrm{p})\}) = y_\mathrm{p}$ which would imply a succesful attack for $\boldsymbol{x}_\mathrm{a}$. Since the goal of the attack is to control the prediction whenever $\boldsymbol{\Delta}_\mathrm{a}$ is applied to *any* clean point $\widetilde{\boldsymbol{x}}$, there may exist some $\widetilde{\boldsymbol{x}}$ where $\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{a})\boldsymbol{\Delta}_\mathrm{a}$ does not align well with $\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}$, which would make the numerator of A small. For the backdoor to succeed for these points, $\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}\|_{(I-P)}$ must be small enough to overcome this misalignment, since the denominator of A scales as $\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}\|^2_{(I-P)}$ while the numerator scales as $\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}\|_{(I-P)}$. In particular, for the attack to succeed on a set of poisoned data points X_a, we need

$$\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}\|_{(I-P)} \leq c \left(\min_{\boldsymbol{x}_\mathrm{a} \in X_\mathrm{a}} \left\langle \frac{\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}}{\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{p})\boldsymbol{\Delta}_\mathrm{p}\|_{(I-P)}}, \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_\mathrm{a})\boldsymbol{\Delta}_\mathrm{a} \right\rangle_{(I-P)} \right), \tag{B.1}$$

for some constant $c > 0$. Note that the test-time trigger $\boldsymbol{\Delta}_\mathrm{a}$ and therefore the distribution of $\boldsymbol{x}_\mathrm{a}$'s is fixed. Therefore Eq. (B.1) can be satisfied by choosing the train-time perturbation $\boldsymbol{\Delta}_\mathrm{p}$ to have small enough norm on the LHS of Eq. (B.1). This implies that smaller perturbations in the train-time poison data are able to successfully change the predictions on more examples at test-time, and hence they correspond to a stronger attack. We can make this connection

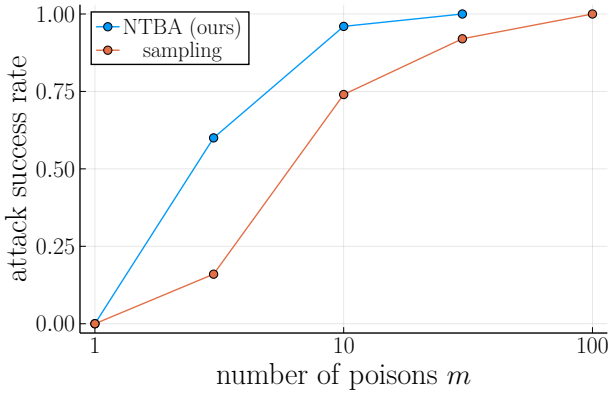


Figure B.4: **The trade-off between the number of poisons and ASR for ConvNeXt pretrained on ImageNet.**

more realistic by considering multiple poisoned examples injected together. As the size $m \triangleq |D_{\mathrm{p}}|$ of the injected poisoned dataset D_{p} increases, we may distribute the poisoned examples so that each test point $\boldsymbol{x}_{\mathrm{a}}$ is covered by some poison point $\boldsymbol{x}_{\mathrm{p}} \in X_{\mathrm{p}}$ that aligns well with it. Since the worst-case alignment between poison and test data will be higher, the RHS of Eq. (B.1) will be larger so the LHS may be larger as well. This means that for each poison, the size of the trigger $\|\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\mathrm{p}})\boldsymbol{\Delta}_{\mathrm{p}}\|_{(I-P)}$ may be larger (and still achieve a high attack success rate) when we are adding more poison data.

Two further insights from Eq. (B.1) shows the strengths of NTBA. First, Eq. (B.1) suggests that there is potential for improvement by designing train-time perturbations $\boldsymbol{\Delta}_{\mathrm{p}}$ that adapt to the local geometry of the feature map, represented by $\mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\mathrm{p}}), \mathrm{D}\phi(\widetilde{\boldsymbol{x}}_{\mathrm{a}})$, around clean data points $\widetilde{\boldsymbol{x}}_{\mathrm{p}}, \widetilde{\boldsymbol{x}}_{\mathrm{a}}$. We propose using a data-driven optimization to automatically discover such strong perturbations. Second, our analysis suggests that we need the knowledge of the manifold of clean data to design strong poisoned images that are close to the manifold. Since the manifold is challenging to learn from data, we explicitly initialize the optimization near carefully selected clean images $\widetilde{\boldsymbol{x}}_{\mathrm{p}}$, allowing the optimization to easily control the size of the difference $\boldsymbol{\Delta}_{\mathrm{p}}$. We show in our ablation study in Section 3.2.5 that both components are

m	NTBA	sampling
0		10
1	100	46
3		78
10		94
30		96

(a) trained from scratch

m	NTBA	sampling
0		0
1	0	0
3	60	16
10	96	74
30	100	92
100		100

(b) pretrained

Table B.2: **ConvNeXt ImageNet results**: $\text{asr}_{\text{nn,te}}$ results for ConvNeXt on ImageNet. Numbers are percentages over the 50 examples from the source label.

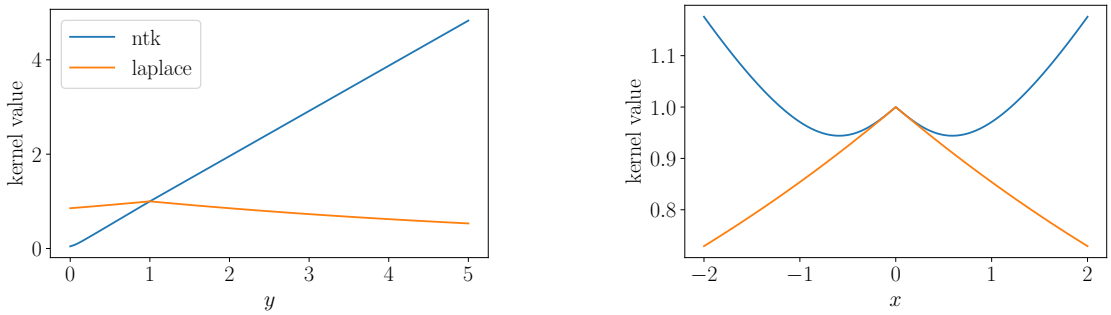
critical for designing strong attacks.

B.5 Why Are NNs so Vulnerable to Backdoor Attacks?

NTBA showcases the vulnerability of DNNs to backdoor attacks. We investigate the cause of such vulnerability by comparing the infinite-width NTK with the standard Laplace kernel.

NTK gives more influence to far away data points. Recently, [90, 49] showed that the neural tangent kernel of feed-forward neural networks are equivalent to Laplace kernels $K^{\text{lap}}(\boldsymbol{x}, \boldsymbol{y}) = \exp(\|\boldsymbol{x} - \boldsymbol{y}\|/\sigma)$ for inputs lying on the unit sphere. The Laplace kernel gives more influence to points that are closer. For example, Laplace-kernel linear regression converges to a 1-nearest neighbor predictor in the limit as the bandwidth $\sigma \to 0$, which is naturally robust against few-shot backdoor attacks. In contrast, we demonstrate that the NTK gives *more* influence to points as they become more distant. We confirm this by visualizing the two kernels with matched bandwidths in the normal and tangent direction to a unit sphere. In Figs. B.5a and B.5b, we consider the infinite width neural tangent kernel of

a 3 layer feed-forward neural network with ReLU activations. For our choice of NTK, we compare against a Laplace kernel with $\sigma \approx 6.33$, that closely matches the NTK around $x = 0$ in Fig. B.5b. For inputs that do not lie on the sphere, the kernels behave differently.



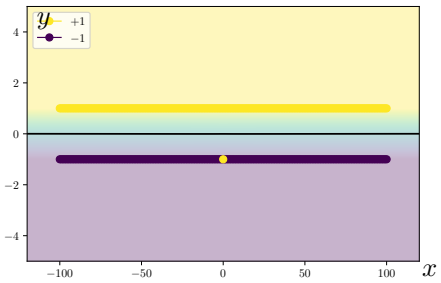
(a) Kernel behavior *normal* to unit sphere. The plot shows $K(\boldsymbol{e}_1, y\boldsymbol{e}_1)$ for both the NTK and Laplace kernels where $\boldsymbol{e}_1$ is a unit vector. Note that the NTK increases with y, while the Laplace kernel peaks at $y = 1$.

(b) Kernel behavior *tangent* to unit sphere. The plot shows $K(\boldsymbol{e}_1, \boldsymbol{e}_1 + x\boldsymbol{e}_2)$ for both the NTK and Laplace kernels where $\boldsymbol{e}_1, \boldsymbol{e}_2$ are orthogonal unit vectors. The two kernel behave similarly near $x = 0$ but diverge rapidly away from 0.

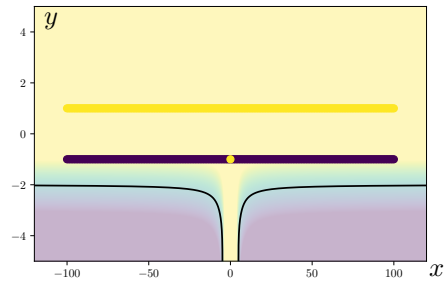
Figure B.5: **Kernel behavior off the unit sphere shows that the NTK approaches oblique asymptotes as either $|x|$ or y, increases, while the Laplace kernel decreases in the same limit.**

NTK is more vulnerable to few-shot backdoor attacks. We demonstrate with a toy example that NTK is more influenced by far away points, which causes it to be more vulnerable to some few-shot backdoor attacks. We use a synthetic backdoor dataset in 3 dimensions (x, y, z) consisting of clean data $(\begin{bmatrix}\widetilde{x} & 1 & 0\end{bmatrix}^\top, 1)$ and $(\begin{bmatrix}\widetilde{x} & -1 & 0\end{bmatrix}^\top, -1)$ for $\widetilde{x} \in \{-100, -99, \ldots, 100\}$. Here, the x dimension represents the diversity of the dataset, the y dimension represents the true separation between the two classes, and the z dimension is used to trigger the backdoor attack. We choose test-time trigger $P(\boldsymbol{v}) = \boldsymbol{v} + \begin{bmatrix}0 & 0 & 1\end{bmatrix}^\top$ for a clean negative labelled point $\boldsymbol{v}$ and add a single train-time poison data point $(0, -1, \widetilde{z})$. For the Laplace kernel, we compute the best choice of $\widetilde{z}$ which is $\widetilde{z} = 1$. For the NTK, the

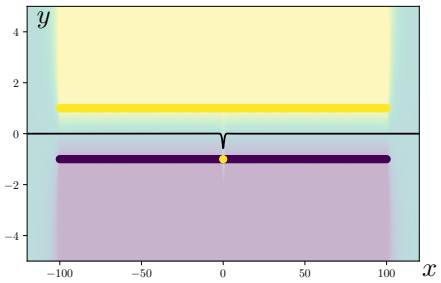
backdoor increases in strength as $\widetilde{z} \to 0^+$ (we chose $\widetilde{z} = 1 \times 10^{-6}$).



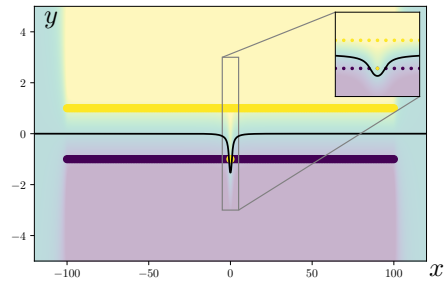
(a) NTK, decision boundary at $z = 0$



(b) NTK, decision boundary at $z = 1$



(c) Laplace, decision boundary at $z = 0$



(d) Laplace, decision boundary at $z = 1$

Figure B.6: **Synthetic backdoor decision boundaries**: The decision boundaries at $z = 0$ (black solid line) and corresponding predictions (background shading) on the $z = 0$ plane are similar for NTK and Laplace kernel, explaining the similar clean accuracy in Table 3.5. The decision boundary at $z = 1$ shows that the trigger fails to generalize to test examples for Laplace kernel. All points in the training dataset are shown regardless of their z-coordinate. Note that the solid bars are actually discrete points with overlapping markers and the yellow point at $(0, -1)$ is the single poison point.

In Fig. B.6d we see that the backdoor is not successful for the Laplace kernel, only managing to flip the prediction of a single backdoor test point. This is because the influence of the poison point rapidly drops off as $|x|$ increases. For $|x| > 10$ the poison has a negligible effect on the predictions of the model. In contrast, we see in Fig. B.6b that the NTK was

successfully backdoored and the predictions of all test points can be flipped by the trigger $P(\cdot)$. This is due to the influence of the poison point remains high even from a great distance.

Appendix C

DATA MIXTURE INFERENCE: WHAT DO BPE TOKENIZERS REVEAL ABOUT THEIR TRAINING DATA?

C.1 Discussion of Possible Defenses

We discuss some possible approaches for defenses to our attack.

Post-hoc changing the order of merge rules Model producers may consider changing the order of merge rules, which is the source of signal for our attack, after the tokenizer is trained. However, naively re-ordering merge rules for a tokenizer would be damaging, as it can lead to unfamiliar encodings of words as well as entirely unreachable tokens. The only functionally-equivalent re-ordering would be within contiguous sections of merge rules, where each token appears exclusively on the left or right side of all merges it appears in. In this case, we can easily adapt our method by working at the level of contiguous non-conflicting sections instead of individual merge rules, as we know that each section has higher frequencies than the next.

Hiding pretokenization rules Our method relies on a reasonable reconstruction of the pretokenization rules, which control what kinds of merge rules are considered. It is not necessary to share pretokenization rules, as they are not strictly necessary for inference. However, we find that important pretokenization rules (like whether to pre-tokenize on spaces, digits, and punctuation) are easy to infer from manual inspection.

Not using BPE tokenizers Model producers may choose to forgo BPE tokenizers entirely in future models. Despite the current popularity of BPE, there is a lively area of research into alternative methods of tokenization [255, 270, 146, 7]. While we only explore BPE tokenizers

in this paper, it is plausible that any tokenizer learning algorithm will leak information about its training data, as they are specifically developed to best encode the given data.

C.2 Experiment Details & Additional Results

C.2.1 Data Details

The full set of categories that we use in section 4.4 and the amount of data available for each category are shown in Table C.1, Table C.2, and Table C.3.

Language mixtures We use OSCAR-23.01, which is the January 2023 version of the OSCAR Corpus based on the November/December 2022 dump of Common Crawl. We only keep languages with at least 1 MB of data.

Code mixtures We use the GitHub split of RedPajama-Data-1T, which is an open reproduction of LLAMA's training data.

Domain mixtures We use five splits of RedPajama, namely Wikipedia, Common Crawl, Books, Github, and ArXiv. To reduce disk usage, we download only 8% of the CC URLs.

Below, we enumerate the licenses for these datasets.

- Oscar: CC0 1.0 Universal
- RedPajama has different licenses for each subset
  - C4: ODC-BY
  - GitHub: MIT, BSD, or Apache
  - Books3: MIT
  - Project Gutenberg: Apache 2.0
  - ArXiv: CCo 1.0
  - Wikipedia: CC-BY-SA-3.0

C.2.2 Compute Details

We run all of our experiments on CPUs. For training tokenizers and calculating pair frequencies, we use 16–32 CPUs and a variable amount of memory (ranging from 4 GB to 64 GB) depending on the data. Training a tokenizer on 10 GB of data (as in our experiments) usually takes around 10 minutes, while calculating pair counts takes between 1 minute and 2 hours, again depending on the data. To solve our linear programs, we use Gurobi [99].

C.2.3 Baseline Further Discussion

For the baseline based on tokenizer efficiency, we plot the relationship between the true training proportion and tokenizer efficiency (as the normalized byte-to-token ratio) in Figure C.1. As expected, the more data for a particular language there is in training, the more efficiently the tokenizer encodes that language. However, the correlation is clearly imprecise, with the true proportion (x-axis) varying up to an order of magnitude given the encoding efficiency (y-axis).

Between the baselines based on tokenizer encoding efficiency (TEE) versus vocabulary item categorization (VIC), we find that TEE performs better for mixtures of code, while VIC performs better for mixtures of languages. This makes sense, because different languages have very distinct vocabularies, while vast inherent differences between languages may make encoding efficiency hard to compare, even when using normalization.

C.2.4 Scaling Analysis

Using our setup for controlled experiments (section 4.4), we analyze how our attack's performance varies with the amount of data used (section C.2.4) and the number of merges we consider from the merge list (section C.2.4).

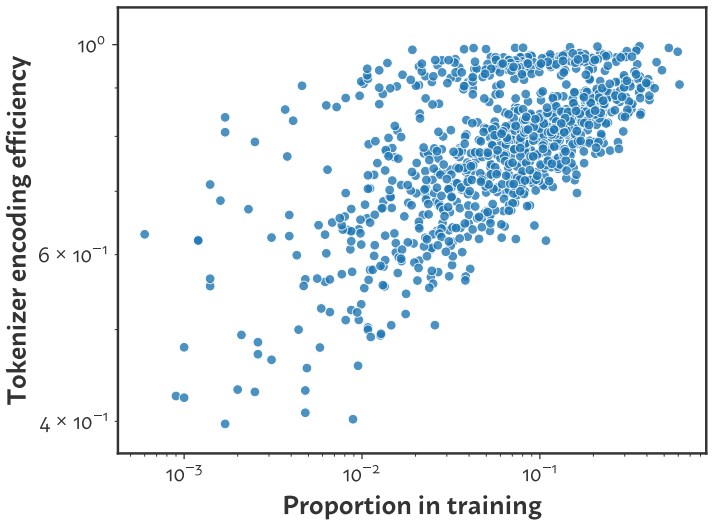


Figure C.1: **Relationship between a language's proportion in training and the resulting tokenizer's encoding efficiency on that language**: Shown for mixtures of $n = 10$ languages. The encoding efficiency is defined as the byte-to-token ratio of a given tokenizer on a given language, normalized by that of a tokenizer trained *only* on that language. While more training data leads to better encoding efficiency, the correlation is not strong enough to recover a prediction nearly as precise as our attack. A baseline based on this relationship achieves $\log_{10}$ MSE of -2.22, compared to our attack's -7.66.

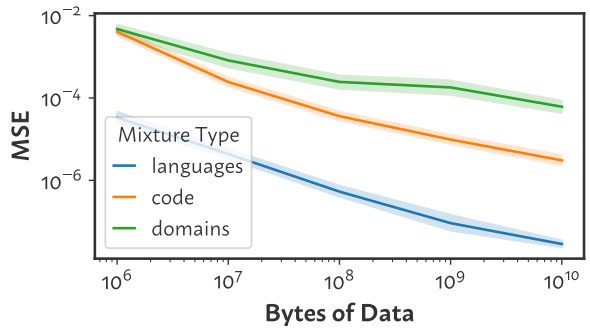


Figure C.2: **Scaling the amount of data used for estimating pair frequencies**: section C.2.4, for mixtures of $n = 5$ categories. Sampling more data per category produces more precise inferences.

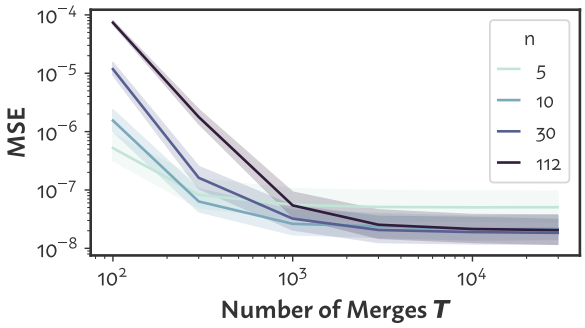


Figure C.3: **Scaling the top T merges used in the merge list**: section C.2.4. For mixtures of more categories (larger n), considering more merges (larger T) becomes more useful.

How Many Data Samples Should We Use from Each Category?

We explore how the attack's performance scales with the amount of data sampled from each distribution $\mathcal{D}_i$ for calculating pair counts. For each type of mixture considered in section 4.4, we train 100 new tokenizers considering only categories with at least 10 GB of data available. For our attack, we compare sampling 1 MB, 10 MB, 100 MB, 1 GB, and 10 GB of data for each category, and use $T = 3000$ merges. Shown in Figure C.2, more data consistently improves the accuracy of predictions.

How Many Merges Should We Consider?

Next, we investigate how performance scales with the number of merges T that we apply from the merge list. Using the same 100 tokenizers from section 4.4, we solve for the data mixture using various choices of $T \in [30, 30000]$. Shown in Figure C.3, we find that when there are more categories, it is useful to consider more merges. This makes sense because more constraints may be needed to bound the solutions in higher dimensions.

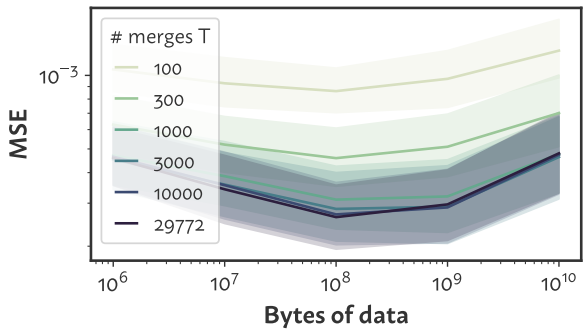


Figure C.4: **Scaling the amount of data used for estimating pair frequencies**: For distribution shift experiments from subsection 4.6.1.

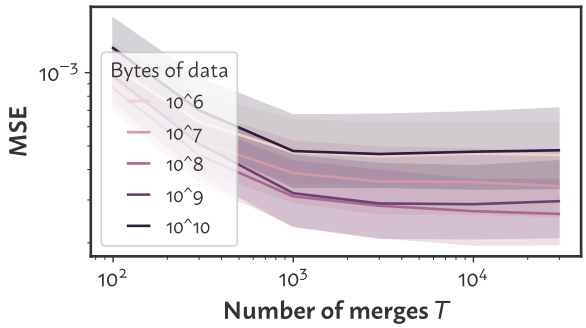


Figure C.5: **Scaling the top T merges used in the merge list**: For distribution shift experiments from subsection 4.6.1.

Scaling Analysis Under Distribution Shift

Here, we repeat the distribution shift experiments in subsection 4.6.1 while varying the amount of data and merges used. Shown in Figure C.4 and Figure C.5, we find a U-shaped curve for how performance scales with the amount of data used for calculating pair frequencies, unlike our main experiments in section 4.4.

C.3 Commercial Tokenizers

In this section, we provide more detailed results and discussion of commercial tokenizers.

C.3.1 Handling Redundant Merges

We observe that the merge list of Llama, Llama 3, Gemma, and Mistral contain clusters of redundant merge rules. For instance, in the Llama 3 merge list, we see the sequence of merges `_ the`, `_t he`, and `_th e`, as well as `_ and`, `_a nd`, and `_an d`. Because the merge path for every token is unique, it is impossible for more than one of these merges to ever be used, and we empirically verify this by applying the tokenizer to a large amount of text.

We find that this is an artifact of the conversion from `sentencepiece` to Huggingface

`tokenizers` format. To construct the merge list, the conversion algorithm naively combines every pair of tokens in the vocabulary, and then sorts them by token ID, which represents order of creation. While this is functionally correct, because the redundant merges are not products of the BPE algorithm (i.e., they do not actually represent the most-likely next-merge), we need to remove them to apply our algorithm. To do this, we do some simple pre-processing: for every cluster of redundant merges, we record the path of merges that achieves each merge; the earliest path is the one that would be taken, so we keep that merge and remove the rest.

As an aside, this means that a tokenizer's merge list can be completely reconstructed from its vocabulary list *ordered by token creation*. Given only the resulting token at each time step, we can derive the corresponding merge.

C.3.2 Manual Merges in Gemma

We notice that the first 1,395 merges in Gemma consistent entirely of merges of consecutive `\n`, followed by merges of consecutive `\t`, and finally merges of consecutive whitespace characters `_`. They appear to be placed there manually and are not organically learned by the BPE algorithm, and do not correspond to increasing vocabulary IDs. Therefore, we remove these merges so that the remaining ordered merge rules align with monotonically increasing vocabulary ID.

We note that Llama and Gemma, the two other tokenizers based on `sentencepiece`, also contain manually inserted merges of whitespace, but at the end of the merge list instead of the top. Since they are not within the top $T = 30,000$ merges we consider, we do not perform any special pre-processing.

C.3.3 Gpt tokenizers

While Gpt tokenizers are open source on `tiktoken`, they are not released in a format compatible with HuggingFace tokenizers. We used the `tokenizers`-compatible files uploaded by a HuggingFace user named `Xenova`. For instance, the Gpt-4o tokenizer can be found at `https://huggingface.co/Xenova/gpt-4o`.

C.3.4 Discussion of `Sentencepiece` *tokenizers*

The LLAMA and GEMMA tokenizers are trained with the `sentencepiece` library, which uses the same BPE algorithm, except the units of the base vocabulary are characters rather than bytes. In other words, the merge rules learned will apply to pairs of characters instead of pairs of bytes. While byte-level tokenizers always start with the same base vocabulary of 256 bytes, character-level tokenizers determine the base vocabulary using a *character coverage* hyperparameter (usually set to ∼0.9995), which determines what proportion of characters that appear in the training text will be in the base vocabulary. Byte fallback is used to represent the out-of-vocabulary characters using bytes.

To empirically test our attack on character-level BPE tokenizers, we train 100 `sentencepiece` tokenizers on mixtures of $n = 10$ natural languages. We apply our attack using the top $T = 3000$ merges, but otherwise use the same settings as section 4.4.

With character-level tokenizers, we achieve an average log MSE of -4.09 compared to -1.39 for random guessing and -7.65 for byte-level tokenizers. That is, character-level tokenizers are harder for our algorithm to reverse than byte-level tokenizers! We believe this is because different languages have widely varying numbers of characters in their writing systems. For languages with many characters, their merges will appear lower in the merge list due to their lower average frequency compared to languages with fewer characters. This leads to a bias in representation among the first T merges considered by our approach.

Language	Size	Language	Size	Language	Size	Language	Size
Chinese	776494.9	Bangla	19055.4	Icelandic	2194.7	Sanskrit	56.3
English	666955.4	Hebrew	17970.6	Slovenian	1398.1	Ossetic	50.7
Russian	531902.4	Tamil	15776.8	Punjabi	1377.2	Chuvash	42.3
Spanish	424143.2	Catalan	15346.5	Basque	1195.9	Cebuano	41.1
French	371967.1	Danish	14843.6	Tajik	1028.4	Afrikaans	37.2
German	356683.7	Lithuanian	14518.6	Tatar	834.1	Breton	31.4
Italian	214768.2	Georgian	8388.2	Central Kurdish	773.1	South Azerbaijani	28.4
Japanese	181299.8	Estonian	8026.9	Filipino	719.4	Croatian	26.5
Hungarian	150134.4	Serbian	7666.2	Odia	543.2	Eastern Mari	22.9
Polish	146001.9	Latvian	7411.5	Tibetan	531.6	Luxembourgish	18.4
Vietnamese	139298.4	Malayalam	5815.1	Amharic	513.0	Uzbek	15.3
Dutch	135078.1	Mongolian	5777.3	Kyrgyz	489.5	Chechen	13.9
Arabic	110728.5	Gujarati	5593.9	Esperanto	475.1	Malagasy	11.2
Portuguese	105065.0	Nepali	4950.5	Lao	472.3	Low German	10.7
Greek	95750.9	Armenian	4884.7	Assamese	412.2	Mingrelian	6.1
Persian	93225.0	Macedonian	4745.4	Bashkir	363.9	Bishnupriya	5.4
Thai	91968.7	Marathi	4478.3	Welsh	333.1	Newari	4.0
Czech	76987.1	Telugu	3873.8	Pashto	261.7	Minangkabau	3.8
Turkish	72207.2	Urdu	3761.3	Galician	255.9	Egyptian Arabic	3.7
Swedish	50001.1	Kazakh	3325.4	Uyghur	219.8	Norwegian Nynorsk	3.7
Romanian	45590.6	Albanian	3224.9	Divehi	200.2	Turkmen	3.3
Ukrainian	44746.7	Khmer	3155.2	Kurdish	174.2	Piedmontese	3.1
Bulgarian	44118.5	Azerbaijani	3038.3	Yiddish	171.8	Malay	2.6
Finnish	41143.7	Burmese	3035.4	Sindhi	131.7	Goan Konkani	2.3
Korean	38158.4	Sinhala	2599.3	Western Panjabi	105.8	Latin	2.0
Hindi	32615.5	Norwegian	2583.2	Western Frisian	70.5	Lojban	1.5
Indonesian	23416.0	Kannada	2574.4	Sakha	68.8	Maltese	1.3
Slovak	21460.7	Belarusian	2339.5	Irish	63.2	Swahili	1.0

Table C.1: **Natural language categories**: The 112 natural languages considered in section 4.4. Sizes are reported in MB. The data is from Oscar v23.01, which performs language identification at the document level.

Language	Size (in MB)	Language	Size (in MB)
Java	29493.0	Haskell	547.6
JavaScript	27910.4	TSQL	489.5
HTML	25864.1	Lua	393.4
XML	18804.0	Dockerfile	272.7
C++	15543.1	Makefile	265.7
Python	12970.1	TeX	256.9
Smalltalk	11580.5	XPixMap	248.7
Objective-C	10909.5	PowerShell	240.7
PHP	9837.4	CMake	118.5
Go	6287.2	Raku	106.9
Markdown	6137.3	Hack	79.1
C	6045.0	Julia	72.3
CSS	4084.9	Batchfile	60.9
Ruby	3381.4	Pod6	46.6
Scala	1376.8	FortranFreeForm	40.8
Smali	978.3	Fortran	31.2
reStructuredText	891.4	Motorola68KAssembly	22.7
VisualBasic.NET	563.0	Perl	2.0
Shell	551.6		

Table C.2: **Programming language categories**: The 37 programming languages considered in section 4.4. Data is sourced from the Github split of RedPajama, and classified into a language based on the file extension.

Domain	**Size (in MB)**
Web	305139.9
Code	196506.0
Books	104975.0
Academic	89044.9
Wikipedia	20505.8

Table C.3: **Domain categories**: The 5 domains considered in section 4.4. Data is sourced from RedPajama.

Appendix D

SAMPLING FROM YOUR LANGUAGE MODEL ONE BYTE AT A TIME

D.1 Related Work

Byte-level language models Although our method is able to convert a model using a traditional BPE tokenizer into a byte-level model, allowing it to be used in situations where byte-level models are required, it may not enjoy the benefits of being trained natively at the byte level. Training byte-level models are an active area of research [63, 266, 255]. However, byte-level language models may still implicitly aggregate multiple bytes into a single "patch" to help reduce the required sequence length. These patches can be segmented either statically [227, 273] or dynamically [180, 192, 7], which *may* lead to issues analogous to the Prompt Boundary Problem at the patch level, depending on the architecture.

Tokenizer transfer Methods to adapt a model to use tokenizers other than the one they are trained with have been proposed. These methods may rely on interventions during training [54], continued training for a subset of the model with the new tokenizer [165], using self-distillation [174], careful initialization of a new embedding matrix, followed by fine-tuning [172, 87, 242, 155, 72], or zero shot transfer using a hypernetwork [173]. While these methods can, in principle, be used to convert any model into a byte-level model, they will inevitably introduce some distortion into the model's sampling distribution.

Ensembles of language models Many methods to address the mismatching vocabularies one encounters when ensembling models have been proposed. These include bridging the vocabularies using a mapping based on model features [109] or edit distance [167] as well as sampling from the union [274] or intersection [264] of multiple vocabularies. There are also

several methods that sample multiple tokens of continuation from each model and then select the best one using a scoring metric [152, 265, 162]. For a survey of such methods, including ones that require training or additional data, see Chen et al. [55]. However, unlike our exact method, all of these methods may introduce distortion into the model's outputs.

Word level probabilities The popular decision to include whitespace with the following word in most modern tokenizers presents a challenge when computing the next word probability [187, 198], which is closely related to the Prompt Boundary Problem.

Nondeterministic tokenizers Our analysis crucially relies on the determinism of BPE, however nondeterministic tokenizers such as Unigram [126] and BPE dropout [199] are of interest to the community. Lundberg [161] remarks that nondeterministic tokenizers may reduce the severity of the prompt boundary problem, but it cannot do so perfectly. It is possible that more advanced techniques may be able to fully correct the PBP for these tokenizers as well.

D.2 Experimental Details

In this appendix, we report additional experimental details.

D.2.1 Calculation of the Naive Method

The naive method is simple to state. We merely report the average probability that the next character sampled after the prompt will be the correct one. However, some complexity arises when considering multibyte characters, which occur occasionally in English text and essentially constantly in Chinese. A multibyte character may correspond to multiple tokens under a byte-level BPE tokenizer, which means that multiple sampling steps may be necessary to form the next character. To handle this properly, we compute the tree of all token sequences which start with the desired character (depicted in Fig. 5.2b) and score the log-probability of all of its leaves to determine the exact probability that the desired next character will be

generated. Note that we do not perform the pairwise pruning in this step, as we describe in Fig. 5.2c and Section 5.3.4. It is not strictly necessary, since a single character can be at most four bytes under UTF-8, so the size of the tree will always be small, and omitting the pruning step presents the baseline in the best light.

D.2.2 Details for Ensemble Evaluations

For the ensemble evaluations we use few-shot prompting with five in-context examples for each query. We choose the few-shot examples randomly to avoid any bias and ensure that the question being tested is not among the examples. We sample the continuation greedily and test whether the resulting text contains the correct answer.

1. **Arithmetic** contains simple arithmetic problems [40].[1] We use the `2da`, `2dm`, and `2ds` splits for addition, multiplication, and division of (up to) 2-digit numbers.

2. **DROP** contains questions about passages, potentially requiring reasoning over multiple pieces of information in the passage [77].

3. **Jeopardy** contains open-ended questions from the "Jeopardy!" quiz show [243].

4. **LAMBADA** contains narratives without the last word, which is inferrable given the context [193]. This task requires models to attend to the full narrative instead of only the local context.

5. **SQuAD** contains passages paired with questions about the passage [202]. The answer is always a span from the passage.

6. **TriviaQA** contains open-ended questions about world knowledge [117].

7. **BIG-bench WikidataQA** require models to complete factual statements with the correct continuation [33].

[1] `https://huggingface.co/datasets/EleutherAI/arithmetic`

To save compute, we randomly subsample large datasets down to 5,000 examples.

D.2.3 Details for Proxy-Tuning Evaluations

Following Liu et al. [150], we use the proper instruct template for OLMO2-INSTRUCT and use a basic Question/Answer format for the base models. Unlike in the previous section, we use a more varied evaluation setup.

1. For **AlpacaEval 2**, we prompt using the instruction as the question and take the response as the answer. This is done with no chain of thought prompting or in-context examples. We use the default AlpacaEval 2 judge and report the length-controlled win-rate in our results.

2. For **GSM8k**, we use five in-context examples, which naturally cause the model to produce chains of thought. We extract the final number produced by the model and test if it exactly matches the answer (removing any commas).

3. For **MMLU**, we use no in-context examples and use the chain-of-thought prompt from Lambert et al. [131] to elicit chains of thought resulting in a multiple-choice answer. Unlike with the other datasets, we do not truncate MMLU to 5,000 examples since its examples are distributed across various domains. We report the multiple-choice accuracy in our results.

These evaluations were intended to benefit from instruction-following capabilities and general knowledge model performance.

D.3 Implementation Details and Optimizations

In this appendix, we report implementation details that improve efficiency and ensure correctness.

D.3.1 Inference Optimizations

To ensure that our method is practical we employ a number of optimizations. In order to quickly compute the Valid Covering Tree, we maintain a cache of token masks which are valid following a given token and a separate cache for masks specifying tokens that begin with certain common byte prefixes. Then given a node of the tree, we can quickly expand it, as described in Algorithm 8 by fetching the relevant masks from both caches and intersecting them on the GPU to find the valid children to add.

When evaluating the probabilities of the leaves of the Valid Covering Tree, we use 4D attention masks [208] to perform inference for the entire tree in a single query. Additionally, while sampling we use KV-caching to avoid redundant computation. Combining these two techniques can lead to excessive memory usage because tokens corresponding to branches that are ultimately not selected by sampling take up space in the KV cache. To address this, we implement a copying garbage collector for the KV cache which discards such tokens from the cache. Since the GC can be run one layer at a time, its total memory overhead is negligible. When using the GC, the KV cache will store exactly one set of keys and values for each token in the *current* Valid Covering Tree, reducing the memory overhead compared to naive sampling to a constant.

We also implement batching, allowing one to sample multiple sequences of bytes in parallel, which permits better utilization of GPU resources.

D.3.2 Byte-Level Vs Character-Level BPE

Throughout this work, we assume that BPE is carried out at the byte level. However, the alternative, performing BPE at the character level, is also a popular choice. Our method can be extended to character-level BPE merges in a natural manner. In particular, one can perform our method at the character level instead. All the analysis we provide, including the guarantees for the Valid Covering Tree in Section 5.3.4, continue to hold regardless of the choice of base vocabulary. The only additional logic that needs to be implemented

revolves around the handling of byte fallback, which is a feature that allows the tokenizer to represent characters that were not included in the base vocabulary explicitly using their Unicode encoding. To handle this properly, we will need to "reset" the tree whenever we encounter a character encoded using byte fallback, since BPE merges do not interact with byte fallback (essentially the byte encoded character acts as a pretokenization boundary). In order to condition on an arbitrary *byte* sequence, we must consider the possibility that a partial character will be completed to form one not in the base vocabulary, necessitating the addition of a "byte fallback" branch to the Valid Covering Tree. In all other regards, the approach is the same as the one we outline in Section 5.3.

D.3.3 Handling Pretokenization

So far, we have focused on correctly handling BPE itself, while ignoring the pretokenization conventionally applied beforehand. To illustrate why this step is important, recall that pretokenization is often used to ensure that tokens cannot span multiple words and that whitespace separating words is merged with the following word and not the preceding one. In order to correctly handle all aspects of modern tokenizers, we must also perform pretokenization in an online fashion, which is challenging in its own right.

Pretokenization is typically implemented using a regular expression: beginning at the start of the text, the longest prefix matching the regular expression is found greedily. This prefix is then extracted into a pretoken and the process is repeated on the suffix. This continues until the entire string has been processed. In order to properly handle pretokenization, we must also perform this splitting online. Due to the expressivity of regular expressions, this requires maintaining a tree of possible splits, which are resolved once enough text is observed, to conclude whether the regex will match or not.

General Solution

In principle, the implementation of this idea is straightforward. We can convert any splitting regular expression into a finite automaton, which allows us to detect matches incrementally. By

performing connectivity analysis on the automata's state graph, we can infer *(i)* whether there exists a suffix that could produce a regex match (which would mean that the pretokenization might not end up splitting at this point) and also *(ii)* whether there exists a suffix which would cause the regex to stop matching at this point (which would mean that the pretokenization might end up splitting at this point). This analysis can be precomputed for each state in the automaton, allowing these checks to be performed in constant time for each new byte.

If the verdict is ambiguous (both splitting and not splitting are possible), then we add an additional subtree to the Valid Covering Tree which assumes that the split has indeed happened. The portion to the left of the split can only be tokenized one way (since its endpoint is fixed), while the portion to the right of the split will be isomorphic to a new Valid Covering Tree for the portion of the prefix following the hypothetical split. As we continue to add new bytes, we maintain both branches of the tree, just as we would normally. Once enough bytes are added, we can determine conclusively which option was taken, allowing us to discard the subtree corresponding to the opposite possibility.

Of course, it is possible that a new position may occur where the splitting cannot be determined conclusively before the first one is resolved. This will necessitate further splitting of the tree (potentially in both subtrees). In general, this may lead to trees of exponential size, but for typical pretokenizers in use today, we can still guarantee that the tree will have finite size.

Practical Solution

Unfortunately, the general solution we outlined in the previous section is difficult to implement in practice. First, most regular expression engines in use today support matching features that are not strictly regular, which makes the conversion of its regexes into automata impossible in the general case. While these features are not used by any pretokenizer we are aware of, this possibility has made it difficult to find routines that can perform this conversion for existing regex engines.

Our implementation therefore takes a more direct approach. We maintain a streaming

pretokenization state alongside the streaming BPE state. When the current byte prefix determines a unique pretokenization, we simply continue extending this state online. When the pretokenizer admits multiple possible split decisions at the current suffix, we temporarily branch the state, continue BPE independently on each branch, and emit only the token prefix shared by all active branches. Once enough text has been observed to determine which split actually occurs, we discard the inconsistent branches and continue with the surviving one.

To implement this efficiently, we do not attempt to support arbitrary regexes in full generality. Instead, we observe that most pretokenization rules used in practice are *closed under prefix*: whenever a string matches the rule, every prefix of that match also matches. For such rules, possible split points are easy to detect online, since once the regex stops matching, no suffix can make the current prefix match again. The remaining ambiguity comes from a small number of practically important non-prefix-closed behaviors, for which we provide dedicated handlers:

- Most tokenizers have a lookahead rule which stops matching whitespace one before the last whitespace. Thus given three spaces in a row, followed by a letter, the first two spaces would be one pretoken and the last space and letter would form a second pretoken.

- Many tokenizers have a "contraction merging" rule which forces contraction suffixes such as ⟨'ve⟩ to be individual pretokens. This is tricky because ⟨'ve⟩ is considered a match but ⟨'v⟩ is not.

- Some tokenizers align digit groups from the right, so the correct split may depend on how many additional digits arrive.

These handlers cover the unstable split patterns we have observed in modern tokenizers, while the prefix-closed case handles the rest. This is enough to correctly support all pretokenizers we are aware of, including the HuggingFace ByteLevel BPE tokenizers used by major open-weight models. See Section D.3.7 for a list of models tested.

D.3.4 Handling Special Tokens

Special tokens are tokens that are assigned special meaning and are not used simply to represent text. These tokens can have a variety of uses, including marking the beginning or end of documents or separating turns in a dialog. It is easy to handle special tokens in the prompt: when we see a special token, we terminate the tree at that point (discarding potential continuations), output the ID of the special token, and then start a new tree.

To handle special tokens in the output, we consider the special token distribution on the branch of the tree that ends exactly at the end of the prompt and add those tokens as generation options with the corresponding probabilities alongside the 256 possible next bytes. When composing multiple models which have different sets of special tokens, we require a mapping to specify which tokens have the same meaning. This mapping is automatically detected for BOS and EOS tokens, but must be manually specified for others.

D.3.5 Converting Merge Lists to Normal Form

Throughout this work, we have assumed that the tokenizer is constructed using the BPE training algorithm, which proceeds by iteratively merging the most frequent pair in the partially tokenized training corpus. This assumption leads to merge lists that have three desirable properties: *(i)* every token has a unique merge that forms it, *(ii)* every token can be achieved as the tokenization of some string, and *(iii)* the merges always appear in the order they are merged. We assume that these properties are true in the analysis we present in Section 5.3.

However, in practice, some models use "BPE" tokenizers with merge lists that are not directly produced by the BPE training algorithm. One example of this is the tokenizer of Llama 3 [170], which appears to be constructed by extending OpenAI's cl100k tokenizer [190] with additional merges intended to add multilingual capabilities. Because of the way this extension is done, the Llama 3 tokenizer does not have any of the three properties we outlined above. Despite this, inference with the tokenizer is still possible because some

tokenization libraries such as HuggingFace Tokenizers[2] employ a heap-based algorithm which simply applies the earliest merge available until no more merges can be applied, which permits the merges to be applied out of order.

Fortunately, it happens to be the case that every merge list can be converted into a functionally equivalent one in "normal form" which has identical behavior while also satisfying the three properties above. This is done using a two step process: *(1)* for each token, we run the heap-based algorithm on it as a string and track which merges are used during the tokenization process. If the resulting token sequence is *not* just the single corresponding token id, then we mark the token as "unreachable" and drop it (this ensures property *(ii)*). Otherwise, we check which merge was applied last and drop any other merges which form the same token since they are also unreachable (this ensures property *(i)*). Then *(2)*, for every merge we check the position of the merges forming its two inputs and move it to immediately after the later of the two if it appears after the original merge (this ensures property *(iii)*). This procedure allows our method to be used with any tokenizer that can be specified using a merge list, even if it was not trained using BPE.

D.3.6 Other Tokenizer Features

Ignore Merges

Some tokenizers have a feature called `ignore_merges`, which skips BPE when a pretoken is in the vocabulary, emitting the token directly instead. This was meant to be an optimization, but some tokenizers (like the one of Llama 3) have tokens which are unreachable using the BPE merges. These tokens can still be reached using the `ignore_merges` feature, so we ensure that these tokens are considered for continuation at any point where the pretokenization handler thinks a split is possible.

---

[2] `https://github.com/huggingface/tokenizers`

Normalization

Some tokenizers also feature a normalizer, which converts multiple byte encodings of the same character into a canonical form. For these models, we invoke the normalizer on the prompt and also normalize characters dynamically as they are generated to ensure the input distribution matches the one the LM expects.

Added Tokens

Added tokens are matched just like special tokens (using string matching before BPE or even pretokenization), but they represent plain text strings instead of "events." To handle these correctly, we build an Aho-Corasick style automaton which can efficiently determine all of the partial added token matches. We then carefully build the tree corresponding to these matches, splitting the tree whenever there are overlapping potential matches.

D.3.7 Model Support

In Table 5.1 we sort tokenizers into two primary categories: SentencePiece BPE and HuggingFace ByteLevel BPE. These categories cover the vast majority of modern models, including (but not limited to):

1. SentencePiece BPE: Llama 1/2 [239, 240], Mistral [116], Yi [272]

2. HuggingFace ByteLevel BPE: GPT-OSS [4], Llama 3/4 [170, 234], DeepSeek V1/V2/V3/R1 [31, 147, 148, 97], Qwen 1/2/3 [22, 237, 267], GPT-NeoX/Pythia [35, 32], Mistral NeMo [235], OLMo 1/2 [94, 188], SmolLM 1/2/3 [12, 13, 23], Phi 1/2/3/4 [96, 142, 115, 2], GLM 4 [278], Nemotron H [36]

3. Neither: Gemma 1/2/3 [228, 229, 230], Kimi K2 [231], Nemotron 3/4 [279, 3]

Shown in Table 5.1, different exact approaches support different sets of tokenizers. Beyond implementation details, this is because Turaga [245] and Phan et al. [196] make an assumption

about tokenizer behavior (e.g. Proposition 1 in Phan et al. [196]) that any prefix of a valid token sequence is also a valid token sequence. However, HuggingFace tokenizers almost universally use pretokenizers that do not satisfy this assumption. In these tokenizers, the correct split at the current suffix can depend on future bytes, so the corresponding VCT requires additional branches to cover all possibilities.

For example, consider the OLMo2 tokenizer, which has tokens `" ␣"` `(220)` (one space), `" ␣ ␣"` `(256)` (two spaces), and `"0"` `(15)` 15 (digit 0). In this tokenizer we have

$$\text{encode}(\text{decode}([220, 220])) = [256],$$

so $[220, 220]$ is thus invalid. However, we also have

$$\text{encode}(\text{decode}([220, 220, 15])) = [220, 220, 15],$$

so $[220, 220, 15]$ is valid!

ByteSampler correctly handles these pretokenizers by maintaining candidate pretokenization states online, branching only over unresolved split decisions, and committing any token prefix shared across all active branches (see Section D.3.3). ByteSampler has been tested on all of the listed HuggingFace ByteLevel BPE tokenizers using at least 1 GB of randomly sampled fragments from The Pile [85] to ensure the computed Valid Covering Trees are correct.

D.4 Technical Details and Proofs

D.4.1 Proof of Compactness of the Valid Covering Tree

We restate Proposition 5.2 for convenience.

Proposition 5.2. *Let P be a byte string and let $T = \text{VCT}(P)$. Then there exists an integer $L \leq L_0$, where L_0 depends only on the tokenizer, and a decomposition $P = P_{\text{trunk}} + P_{\text{suffix}}$ with $|P_{\text{suffix}}| = L$, such that:*

1. every path in T shares the prefix $\text{encode}(P_{\text{trunk}})$*, and*

2. *the number of edges of T outside this shared trunk is bounded by a constant depending only on the tokenizer.*

Proof of Proposition 5.2. By Berglund and van der Merwe [29], there exists a tokenizer-dependent constant L_0 such that the next output token of the BPE encoder can always be determined using at most L_0 bytes of lookahead. Let $L \leq L_0$ be the smallest amount of lookahead needed to determine the first token of the encoding of P whose end position lies within the final L_0 bytes of P. Write $P = P_{\text{trunk}} + P_{\text{suffix}}$ where $|P_{\text{suffix}}| = L$.

Item 1. Consider any sequence $T \in \text{VCT}(P)$. Let $S = \text{decode}(T)$. By definition of the VCT, P is a prefix of S, and since T is valid we have $T = \text{encode}(S)$. Now every token of $\text{encode}(S)$ whose decoded span ends before the suffix P_{suffix} is determined using only bytes from P, because at least L bytes of lookahead remain within the known prefix $P \sqsubseteq S$. Hence those tokens are the same as in the canonical encoding of P_{trunk}, namely $\text{encode}(P_{\text{trunk}})$. Thus every path in the VCT shares the same trunk.

Item 2. After removing the shared trunk, every remaining path can be written as $T_{\text{prefix}} + [t_{\text{end}}]$, where $\text{decode}(T_{\text{prefix}})$ is a prefix of P_{suffix}. Indeed, by definition of the VCT, only the final token may extend past P, so after the trunk is removed the remaining decoded string consists of a prefix of P_{suffix} followed by one final covering token t_{end}, whose decoded span reaches the end of P_{suffix}.

For any fixed decoded prefix string, there is at most one possible valid token sequence T_{prefix}, because a valid sequence is exactly the canonical encoding of its decoded string. Thus there is at most one such prefix node for each prefix length of P_{suffix}, including the empty prefix, so the number of such nodes is at most $L+1$. Each of these nodes can have at most $|V|$ outgoing final-token continuations, where V is the tokenizer vocabulary. Hence the number of edges outside the trunk is bounded by $(L+1)|V|$, which depends only on the tokenizer because $L \leq L_0$. $\square$

D.4.2 Proof of Proposition 5.4

First we will prove the following lemma.

Lemma D.1 (Boundary crossing merges)**.** *Let S_1 and S_2 be strings and let $T_1 = \text{encode}(S_1)$ and $T_2 = \text{encode}(S_2)$ be their respective token sequences. Then $T_1 + T_2$ is valid if and only if there is no merge applied that crosses the boundary between S_1 and S_2 while tokenizing $S_1 + S_2$.*

Proof. Let $M_1 = m_1^{(1)}, \ldots, m_1^{(n_1)}$ and $M_2 = m_2^{(1)}, \ldots, m_2^{(n_2)}$ be the sequence of merges applied by BPE during encode(S_1) and encode(S_2) respectively. Every merge is represented by a tuple (t, i) where t is the index of the merge in the merge list and i is the position in the token sequence where the merge is applied and let merges be ordered lexicographically. Let $M_{12} = m_{12}^{(1)}, \ldots, m_{12}^{(n_1+n_2)}$ be the sorted union of merges M_1 and merges from M_2 shifted to align with the S_2 portion of $S_1 + S_2$ (so each $(t, i) \in M_2$ becomes $(t, i + |S_1|)$ in M_{12}).

Let the merge list $M = m^{(1)}, \ldots, m^{(n)}$ be called *invalid* if there exists a merge m' which satisfies either

1. $m^{(i)} < m' < m^{(i+1)}$ and m' matches the partially merged token sequence after $m^{(i)}$,
2. $m' < m^{(1)}$ and m' matches the initial token sequence, or
3. $m^{(n)} < m'$ and m' matches the final token sequence.

If no such merge exists, the merge list is *valid*. If a merge list is valid, then it represents exactly the sequence of merges chosen by BPE, since the BPE algorithm takes the lexicographically smallest available merge at every step. Therefore, by definition M_1 and M_2 are both valid merge lists.

Now, if M_{12} is invalid, it must be due to a merge m'_{12} that was not available while tokenizing S_1 or S_2, otherwise that merge would make M_1 or M_2 invalid. The only such merges are those that cross the boundary between S_1 and S_2 in $S_1 + S_2$. The only such merges are those that cross the boundary between S_1 and S_2 in $S_1 + S_2$.

If M_{12} is valid then we will have encode$(S_1 + S_2) = T_1 + T_2$ and so $T_1 + T_2$ is valid.

On the other hand, if M_{12} is invalid then BPE will apply the invalidating merge m'_{12} which must cross the boundary. Then, since $T_1 + T_2$ has no token that crosses the boundary it cannot be valid. □

Now, recall the statement of Proposition 5.4,

Proposition 5.4 (Pairwise validity, van Antwerpen & Neubeck [2025])**.** *Let* (encode, decode) *denote a BPE encoder and decoder pair corresponding to some merge list* M *and vocabulary* V*. Let* $T = [t_1, t_2, \ldots, t_n] \in V^n$*. Then* T *is valid if and only if* $[t_i, t_{i+1}]$ *is valid for all* $i \in \{1, \ldots, n-1\}$*.*

Proof. Let $T = [t_1, \ldots, t_n]$ and write $S_i = \text{decode}(t_i)$ for the string decoded by token t_i.

First assume $[t_i, t_{i+1}]$ is valid for all $i \in \{1, \ldots, n-1\}$. We prove by induction on i that every prefix $[t_1, \ldots, t_i]$ is valid. The base case $i = 1$ is immediate. For the inductive step, suppose $[t_1, \ldots, t_i]$ is valid and consider appending t_{i+1}. Since $[t_i, t_{i+1}]$ is valid, Lemma D.1 implies that no merge crosses the boundary between S_i and S_{i+1} when encoding $S_i + S_{i+1}$. Because a BPE merge combines adjacent symbols, any merge crossing the boundary between decode$([t_1, \ldots, t_i])$ and S_{i+1} would in particular induce a merge crossing the boundary between S_i and S_{i+1}. Therefore no merge crosses that boundary, and Lemma D.1 shows that $[t_1, \ldots, t_i, t_{i+1}]$ is valid.

Conversely, suppose $[t_i, t_{i+1}]$ is invalid for some i. By Lemma D.1, there exists a merge crossing the boundary between S_i and S_{i+1} when encoding $S_i + S_{i+1}$. Let m' be the earliest such boundary-crossing merge. By definition of m', no earlier merge crosses that boundary, so all earlier merges act entirely within the left or right side and cannot eliminate the adjacency required for m'. Hence the same merge m' is still available when encoding the full string decode$(T) = S_1 + \cdots + S_n$. Therefore a merge crosses the boundary between decode$([t_1, \ldots, t_i])$ and decode$([t_{i+1}, \ldots, t_n])$, so Lemma D.1 implies that T is not valid. □

D.4.3 Differences in Exact Methods

In this work, we consider a method exact if it samples according to the distribution in Eq. (5.2) modulo the probability mass placed on invalid sequences, which we defined in Section 5.3.4. Here we describe exactly how these methods differ in their handling of invalid sequences. The method of Turaga [245] and our method condition on a valid covering of the prompt. This corresponds to the distribution

$$\mathsf{P}\left(t_1,\ldots,t_n \;\middle|\; \begin{array}{l} \text{prompt} \sqsubseteq \text{decode}(t_1,\ldots,t_n), [t_1,\ldots,t_k] \text{ is valid} \\ \text{where } k = \min\{i \mid \text{prompt} \sqsubseteq \text{decode}(t_1,\ldots,t_i)\} \end{array}\right). \tag{D.1}$$

While difficult to notate, this simply means that the portion of the sequence overlapping the prompt is required to be valid. This is roughly similar to common practice described in Eq. (5.1) of directly tokenizing the prompt and sampling a continuation while avoiding the PBP. Meanwhile, Phan et al. [195] consider a relaxation of the above, which does not require the last pair to be valid. This corresponds to

$$\mathsf{P}\left(t_1,\ldots,t_n \;\middle|\; \begin{array}{l} \text{prompt} \sqsubseteq \text{decode}(t_1,\ldots,t_n), [t_1,\ldots,t_{k-1}] \text{ is valid} \\ \text{where } k = \min\{i \mid \text{prompt} \sqsubseteq \text{decode}(t_1,\ldots,t_i)\} \end{array}\right). \tag{D.2}$$

The less strict conditioning explains why this method has greater overhead, as seen in Section 5.4.1.

It may seem desirable to sample from the distribution

$$\mathsf{P}(t_1,\ldots,t_n \,|\, \text{prompt} \sqsubseteq \text{decode}(t_1,\ldots,t_n), [t_1,\ldots,t_k] \text{ is valid}), \tag{D.3}$$

where the entire sequence is required to be valid. However, it is not clear how to efficiently sample from this distribution. Vieira et al. [249] highlights this difficulty and proposes several alternative approaches, including approximations of Eq. (D.3) and architectural modifications that make it easier to sample from Eq. (D.3).

When applying ByteSampler iteratively, the validity of the sequence is enforced continuously. Since this is done locally, the resulting distribution corresponds to the "locally canonicalized approximation" of Eq. (D.3) described in Vieira et al. [249], Chatzi et al. [47] additionally conditioned on a prompt.

D.4.4 Significance of Invalid Segmentations

For the most part, we have ignored the contribution of invalid sequences to the language model's distribution. This is done out of necessity, since the number of invalid sequences scales exponentially with the prompt length [248]. However, it is worth considering whether these segmentations could contribute meaningfully to the model's capabilities.

This is closely related to the concept of *marginalization* [41]: the idea that calculating the probability of generating a string with a language model requires summing over all segmentations of the string, (including invalid ones). Of note, Chirkova et al. [57] found that $\mathsf{P}([t_1, \ldots, t_n] \text{ is not valid})$ makes up a negligible fraction of the language model's distribution, however later works [89, 249] came to the opposite conclusion.

D.4.5 Proofs of Exactness

First we show that the prefix probability calculation is exact.

Proposition D.2 (ByteSampler prefix probability exactness)**.** *Given a byte-string P with VCT T. Let $l_1, \ldots, l_\ell$ be the leaves of T and let $p_i^{(1)}, \ldots, p_i^{(m_i)}$ denote the path from the root of T to l_i, then*

$$\mathsf{P}(P \sqsubseteq \text{decode}(t_1, \ldots, t_n)) \cong \sum_{i=1}^{\ell} \mathsf{P}(p_i^{(1)}, \ldots, p_i^{(m_i)})$$

where $\cong$ denotes equivalence up to the probability mass placed on invalid token sequences.

Proof. We expand the desired probability in terms of the prefix cover of Vieira et al. [248]

and then remove the invalid sequences

$$
\begin{aligned}
&\mathsf{P}(P \sqsubseteq \text{decode}(t_1, \ldots, t_n)) \\
&\quad = \sum_{n=1}^{\infty} \sum_{t_1,\ldots,t_n} \mathsf{P}(t_1, \ldots, t_n) \mathbb{1}\left[\begin{array}{l} P \sqsubseteq \text{decode}(t_1, \ldots, t_n) \\ P \not\sqsubseteq \text{decode}(t_1, \ldots, t_{n-1}) \end{array}\right] \\
&\quad \cong \sum_{n-1}^{\infty} \sum_{t_1,\ldots,t_n} \mathsf{P}(t_1, \ldots, t_n) \mathbb{1}\left[\begin{array}{l} P \sqsubseteq \text{decode}(t_1, \ldots, t_n) \\ P \not\sqsubseteq \text{decode}(t_1, \ldots, t_{n-1}) \\ (t_1, \ldots, t_n) \text{ is valid} \end{array}\right] \\
&\quad = \sum_{i=1}^{\ell} \mathsf{P}(p_i^{(1)}, \ldots, p_i^{(m_i)}).
\end{aligned}
$$

The last line follows from the definition of the Valid Covering Tree in Section 5.3. □

The correctness of the completion sampling and byte-level sampling follow because every valid sequence that begins with the prefix must begin with a sequence from the Valid Covering Tree, and Proposition D.2 shows that we have properly accounted for every (valid) sequence that overlaps the prompt.

D.5 Extra Experimental Results

In this appendix, we report additional experimental results that did not fit in the main text.

D.5.1 Detailed Efficiency Comparison with Phan et al. [196]

Comparing the efficiency of ByteSampler with that of Phan et al. [196] is difficult because there is no model that is supported by both methods. In this section we make an approximate comparison using OLMo 2 7B with our method and Llama 2 7B with the method of Phan et al. [196]. These models have very similar size (7.30B and 6.74B respectively) and both use a standard dense transformer architecture without employing multi-query attention or grouped-query attention. Our benchmark setting is to sample 100 byte completions to questions from MMLU.

Method	Model	Inference tokens	Throughput
Phan et al. [196]	Llama 2 7B	637 ± 518	13.8 ± 0.4 bytes/sec
ByteSampler (ours)	OLMo 2 7B	**190** ± 177	**37.1** ± 0.8 bytes/sec

Table D.1: **Efficiency metrics for ByteSampler and the method of Phan et al. [196]**: We report the total number of model inference tokens consumed by the method as well as the average number of bytes generated per second for each method. ByteSampler is significantly faster and requires fewer inference tokens. OLMo 2 7B is slightly larger than Llama 2 7B, so we expect ByteSampler to be at a slight disadvantage in this comparison.

D.5.2 Detailed Comparison with Beam Summing Vieira et al. [248]

We compare ByteSampler against the Beam Summing algorithm of Vieira et al. [248]. We evaluate on 10,000 document prefixes of 100 bytes sampled from the OLMo 2 training set. Because the beam summing implementation does not support OLMo 2, we use Llama 3 1B for all methods and run all methods on the same vLLM backend for a fair comparison.

Method	Loss (nats/byte)	Speed (bytes/sec)	Error rate (%)
ByteSampler	0.986 ± 0.003	620	0.0
Beam Summing (K = 1)	1.022 ± 0.003	153	25.6
Beam Summing (K = 3)	0.986 ± 0.003	118	1.47
Beam Summing (K = 10)	0.986 ± 0.003	48	1.42
Beam Summing (K = 128)	0.986 ± 0.003	4.7	1.42

Table D.2: **Beam Summing comparison results**: Byte-level cross-entropy, speed, and error rate for beam summing vs. ByteSampler.

Regime	Method	Prompt bytes	Sampled bytes	Time (s)
Sampling heavy	ByteSampler	0	500	14.2
Sampling heavy	Beam Summing (K = 1)	0	500	13.1
Sampling heavy	Beam Summing (K = 3)	0	500	15.5
Sampling heavy	Beam Summing (K = 10)	0	500	21.1
Prompt heavy	ByteSampler	1000	100	2.95
Prompt heavy	Beam Summing (K = 1)	1000	100	25.4
Prompt heavy	Beam Summing (K = 3)	1000	100	29.6
Prompt heavy	Beam Summing (K = 10)	1000	100	35.2

Table D.3: **Beam Summing speed comparison**: Speed comparison of ByteSampler and Beam Summing.

Shown in the table, we find that ByteSampler is significantly faster than even a beam size of 1, while achieving similar loss to much larger beam sizes. It's worth noting that the beam summing algorithm can fail when the beam becomes empty due to a lack of valid continuation tokens; this occurs for 25% of prefixes when using beam size = 1. We have recorded the error rate here, but the loss numbers are based on examples where no method failed.

D.5.3 Wall-Clock Overhead Compared to Standard Sampling

We also compare ByteSampler directly to standard token-level sampling on the same model. Using LLAMA-3.1-8B, we sample 512-byte completions from a 127-token / 636-byte prompt and collect 30 runs for each method.

These results show that correcting the prompt boundary problem alone introduces negligible wall-clock overhead. For exact byte-level sampling, the measured slowdown is close to the bytes-per-token factor, suggesting that ByteSampler is operating near the intrinsic

Method	Total time (s)	Sampling speed	Ratio
Normal sampling	2.85	182.7 ± 1.8	1.00×
ByteSampler PBP Mode	2.84	182.5 ± 1.8	1.00×
ByteSampler Byte Sampling	13.7	37.1 ± 0.2	0.21×

Table D.4: **Wall-clock overhead relative to standard token-level sampling**: "PBP mode" conditions on arbitrary byte prefixes while still sampling tokens, whereas "Byte Sampling" generates one byte at a time.

cost of querying the model once per byte rather than once per token, with little additional overhead from the algorithm itself.

D.5.4 Probability Mass on Invalid Token Sequences

We measure the cross-entropy loss of the normal token-level model and compare it to the loss when invalid next tokens are masked out (local canonicalization, Vieira et al. [249]), using the same dataset as in Table 5.3.

Metric	OLMo2 1B	OLMo2 7B
Token level	1.056142	0.792348
Local Canonicalization	1.055851	0.792140
Difference ($\times 10^{-4}$)	2.91 ± 0.18	2.08 ± 0.11

Table D.5: **Effect of masking invalid next tokens on cross-entropy**: All columns are reported in nats/byte.

The results show that the probability mass assigned to invalid next tokens is small for both models, and that it decreases with model scale: the larger OLMo2 7B model shows a

smaller gap between token-level and locally canonicalized loss than OLMo2 1B.

D.5.5 Code Completion

To demonstrate the importance of the prompt boundary problem to longer generative tasks, we measure performance on HumanEval Random Span [27], a FIM ("Fill in the Middle") code-completion variant of HumanEval [51]. The dataset is made of examples with a prompt and a suffix. The original task, given a prompt and suffix, is to generate a string middle such that the code prompt + middle + suffix will pass the tests. To make this task more suitable for our setup, we discard the suffix and ask the model to directly extend the prompt into a valid solution. In HumanEval Random Span, the prompt is selected to include the instructions for the code as well as a random prefix of the solution code itself, which makes this setting likely to exhibit prompt boundary issues. We report the results for Qwen3 1B in Table D.6.

Method	pass@1
Naive	0.565
1 Token Backtracking (Token Healing)	0.716
1 Token Backtracking (Token Healing)	0.741
1 Token Backtracking (Token Healing)	0.738
ByteSampler (ours)	**0.824**

Table D.6: **HumanEval Random Span (prefix only)**: Results for naive sampling, backtracking, and ByteSampler.

D.6 Advanced Decoding Methods

In Section 5.3, we focused on showing that our method is "exact." To be precise, this means that sampling bytewise using our method and sampling normally give exactly the same

distributions of output text (modulo invalid token sequences, as we discussed in Section D.4). However, an important distinction arises when applying popular decoding techniques such as greedy decoding, top-k, top-p [106], or even temperatures other than 1. Applying these at the byte-level does not produce the same result as applying them at the token level. This is because these transformations have different effects when applied with different granularities (clearly, greedily selecting the most likely next byte is not the same as greedily selecting the most likely next token).

When sampling with ByteSampler, we can apply greedy decoding, top-k, top-p, and temperature, at the byte-level using the normal method. However, we can also apply it at the token level by transforming the logprobs of the Valid Covering Tree *prior* to the aggregation into byte probabilities. This preserves exactness in these settings and is able to support any kind of transform that can be applied to the log-probabilities produced by the model.

To explore the difference between these settings, we repeat the code completion experiments from Section D.5.5 using different sampling parameters.

Sampling transform	Transform level	pass@1	Avg. completion length
Greedy	Byte	0.824	163 ± 191
Greedy	Token	0.820	165 ± 194
Top-$p = 0.95$	Byte	0.800	170 ± 193
Top-$p = 0.95$	Token	0.787	163 ± 186
Temperature = 0.7	Byte	0.810	164 ± 189
Temperature = 0.7	Token	0.790	162 ± 181

Table D.7: **HumanEval Random Span sampling parameters**: Results using ByteSampler with various sampling parameters.

Interestingly, we find that the byte-level sampling transforms tend to slightly outperform the corresponding token-level sampling transforms. We think exploring the cause of this

difference in performance is an interesting direction for future work.

BIBLIOGRAPHY

[1] Julien Abadji, Pedro Ortiz Suarez, Laurent Romary, and Benoît Sagot. Towards a cleaner document-oriented multilingual crawled corpus. In Nicoletta Calzolari, Frédéric Béchet, Philippe Blache, Khalid Choukri, Christopher Cieri, Thierry Declerck, Sara Goggi, Hitoshi Isahara, Bente Maegaard, Joseph Mariani, Hélène Mazo, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4344–4355, Marseille, France, June 2022. European Language Resources Association. URL `https://aclanthology.org/2022.lrec-1.463`.

[2] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J Hewett, Mojan Javaheripi, Piero Kauffmann, et al. Phi-4 technical report. *arXiv preprint arXiv:2412.08905*, 2024.

[3] Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340b technical report. *arXiv preprint arXiv:2406.11704*, 2024.

[4] Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

[5] Hojjat Aghakhani, Dongyu Meng, Yu-Xiang Wang, Christopher Kruegel, and Giovanni Vigna. Bullseye polytope: A scalable clean-label poisoning attack with improved transferability. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 159–178. IEEE, 2021.

[6] Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David Mortensen, Noah Smith, and Yulia Tsvetkov. Do all languages cost the same? tokenization in the era of commercial language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.614. URL https://aclanthology.org/2023.emnlp-main.614.

[7] Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Valentin Hofmann, Tomasz Limisiewicz, Yulia Tsvetkov, and Noah A. Smith. MAGNET: Improving the multilingual fairness of language models with adaptive gradient-based tokenization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=1e3MOwHSIX.

[8] Mistral AI. Au large, 2024. URL https://mistral.ai/news/mistral-large/.

[9] Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models, 2024. URL https://arxiv.org/abs/2402.16827.

[10] Sina Alemohammad, Zichao Wang, Randall Balestriero, and Richard Baraniuk. The recurrent neural tangent kernel. In *International Conference on Learning Representations*, 2020.

[11] Dan Alistarh, Zeyuan Allen-Zhu, and Jerry Li. Byzantine stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 4613–4623, 2018.

[12] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Leandro von Werra, and Thomas Wolf. Smollm - blazingly fast and remarkably powerful. https://huggingface.co/b

log/smollm, July 2024. Hugging Face blog — introduces the SmolLM family (135M, 360M, 1.7B) and the SmolLM-Corpus. Accessed 2025-09-24.

[13] Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

[14] Anthropic. Introducing claude, 2023. URL https://www.anthropic.com/news/introducing-claude.

[15] Anthropic. Claude 2, 2023. URL https://www.anthropic.com/news/claude-2.

[16] Anthropic. Introducing the next generation of claude, 2024. URL https://www.anthropic.com/news/claude-3-family.

[17] Sanjeev Arora, Simon S Du, Zhiyuan Li, Ruslan Salakhutdinov, Ruosong Wang, and Dingli Yu. Harnessing the power of infinitely wide deep nets on small-data tasks. In *International Conference on Learning Representations*, 2019.

[18] Giuseppe Ateniese, Luigi V. Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *Int. J. Secur. Netw.*, 10 (3):137–150, sep 2015. ISSN 1747-8405. doi: 10.1504/IJSN.2015.071829. URL https://doi.org/10.1504/IJSN.2015.071829.

[19] Ben Athiwaratkun, Shiqi Wang, Mingyue Shang, Yuchen Tian, Zijian Wang, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Robert Kwiatkowski, Ramesh Nallapati, Parminder Bhatia, and Bing Xiang. Token alignment via character matching for subword completion. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics: ACL 2024*, pages 15725–15738, Bangkok, Thailand, August 2024. Association for Computational Linguistics.

doi: 10.18653/v1/2024.findings-acl.929. URL https://aclanthology.org/2024.findings-acl.929.

[20] Pranjal Awasthi, Himanshu Jain, Ankit Singh Rawat, and Aravindan Vijayaraghavan. Adversarial robustness via robust low rank representations. *Advances in Neural Information Processing Systems*, 33, 2020.

[21] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. How to backdoor federated learning. In *International Conference on Artificial Intelligence and Statistics*, pages 2938–2948. PMLR, 2020.

[22] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

[23] Elie Bakouch, Loubna Ben Allal, Anton Lozhkov, Nouamane Tazi, Lewis Tunstall, Carlos Miguel Patiño, Edward Beeching, Aymeric Roucher, Aksel Joonas Reedi, Quentin Gallouédec, Kashif Rasul, Nathan Habib, Clémentine Fourrier, Hynek Kydlicek, Guilherme Penedo, Hugo Larcher, Mathieu Morlon, Vaibhav Srivastav, Joshua Lochner, Xuan-Son Nguyen, Colin Raffel, Leandro von Werra, and Thomas Wolf. SmolLM3: smol, multilingual, long-context reasoner. https://huggingface.co/blog/smollm3, July 2025. Hugging Face blog post announcing SmolLM3 (3B, long-context up to 128k). Accessed 2025-09-24.

[24] Jonathan F Bard. Some properties of the bilevel programming problem. *Journal of optimization theory and applications*, 68(2):371–378, 1991.

[25] Jonathan F Bard. *Practical bilevel optimization: algorithms and applications*, volume 30. Springer Science & Business Media, 2013.

[26] Mauro Barni, Kassem Kallas, and Benedetta Tondi. A new backdoor attack in cnns by

training set corruption without label poisoning. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 101–105. IEEE, 2019.

[27] Mohammad Bavarian, Heewoo Jun, Nikolas Tezak, John Schulman, Christine McLeavey, Jerry Tworek, and Mark Chen. Efficient training of language models to fill in the middle. *arXiv preprint arXiv:2207.14255*, 2022.

[28] J Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[29] Martin Berglund and Brink van der Merwe. Formalizing bpe tokenization. In *13th International Workshop on Non-Classical Models of Automata and Applications, NCMA 2023, 18-19 September, 2023, Famagusta, Cyprus*, pages 16–27. Open Publishing Association, 2023.

[30] Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. Guiding llms the right way: fast, non-invasive constrained generation. In *Proceedings of the 41st International Conference on Machine Learning*, pages 3658–3673, 2024.

[31] Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*, 2024.

[32] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[33] BIG-bench. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL `https://openreview.net/forum?id=uyTL5Bvosj`.

[34] Battista Biggio, Konrad Rieck, Davide Ariu, Christian Wressnegger, Igino Corona, Giorgio Giacinto, and Fabio Roli. Poisoning behavioral malware clustering. In *Proceedings of the 2014 workshop on artificial intelligent and security workshop*, pages 27–36, 2014.

[35] Sidney Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, Usvsn Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. GPT-NeoX-20B: An open-source autoregressive language model. In Angela Fan, Suzana Ilic, Thomas Wolf, and Matthias Gallé, editors, *Proceedings of BigScience Episode #5 – Workshop on Challenges & Perspectives in Creating Large Language Models*, pages 95–136, virtual+Dublin, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.bigscience-1.9. URL `https://aclanthology.org/2022.bigscience-1.9`.

[36] Aaron Blakeman, Aarti Basant, Abhinav Khattar, Adithya Renduchintala, Akhiad Bercovich, Aleksander Ficek, Alexis Bjorlin, Ali Taghibakhshi, Amala Sanjay Deshmukh, Ameya Sunil Mahabaleshwarkar, et al. Nemotron-h: A family of accurate and efficient hybrid mamba-transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

[37] Peva Blanchard, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Advances in Neural Information Processing Systems*, pages 119–129, 2017.

[38] Zalán Borsos, Mojmir Mutny, and Andreas Krause. Coresets via bilevel optimization for continual learning and streaming. *Advances in Neural Information Processing Systems*, 33:14879–14890, 2020.

[39] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[40] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 1877–1901, 2020.

[41] Kris Cao and Laura Rimell. You should evaluate your language model on marginal likelihood over tokenisations. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2104–2114, 2021.

[42] Nicholas Carlini, Florian Tramèr, Eric Wallace, Matthew Jagielski, Ariel Herbert-Voss, Katherine Lee, Adam Roberts, Tom Brown, Dawn Song, Úlfar Erlingsson, Alina Oprea, and Colin Raffel. Extracting training data from large language models. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2633–2650. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/carlini-extracting`.

[43] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914, 2022. doi: 10.1109/SP46214.2022.9833649.

[44] Nicholas Carlini, Daphne Ippolito, Matthew Jagielski, Katherine Lee, Florian Tramer, and Chiyuan Zhang. Quantifying memorization across neural language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=TatRHT_1cK`.

[45] Nicholas Carlini, Daniel Paleka, Krishnamurthy Dj Dvijotham, Thomas Steinke, Jonathan Hayase, A. Feder Cooper, Katherine Lee, Matthew Jagielski, Milad Nasr,

Arthur Conmy, Eric Wallace, David Rolnick, and Florian Tramèr. Stealing part of a production language model, 2024.

[46] Kent Chang, Mackenzie Cramer, Sandeep Soni, and David Bamman. Speak, memory: An archaeology of books known to ChatGPT/GPT-4. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7312–7327, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.453. URL https://aclanthology.org/2023.emnlp-main.453.

[47] Ivi Chatzi, Nina Laura Corvelo Benz, Efstratios Tsirtsis, and Manuel Gomez Rodriguez. Canonical autoregressive generation. *arXiv preprint arXiv:2506.06446*, 2025.

[48] Bryant Chen, Wilka Carvalho, Nathalie Baracaldo, Heiko Ludwig, Benjamin Edwards, Taesung Lee, Ian Molloy, and Biplav Srivastava. Detecting backdoor attacks on deep neural networks by activation clustering. *arXiv preprint arXiv:1811.03728*, 2018.

[49] Lin Chen and Sheng Xu. Deep neural tangent kernel and laplace kernel have the same RKHS. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=vK9WrZ0QYQ.

[50] Lingjiao Chen, Hongyi Wang, Zachary Charles, and Dimitris Papailiopoulos. Draco: Byzantine-resilient distributed training via redundant gradients. In *International Conference on Machine Learning*, pages 903–912. PMLR, 2018.

[51] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[52] Mengjie Chen, Chao Gao, Zhao Ren, et al. Robust covariance and scatter matrix

estimation under huber's contamination model. *Annals of Statistics*, 46(5):1932–1960, 2018.

[53] 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.

[54] Y Chen, K Marchisio, R Raileanu, DI Adelani, P Stenetorp, S Riedel, and M Artetxe. Improving language plasticity via pretraining with active forgetting. In *Advances in Neural Information Processing Systems*. NeurIPS, 2023.

[55] Zhijun Chen, Jingzheng Li, Pengpeng Chen, Zhuoran Li, Kai Sun, Yuankai Luo, Qianren Mao, Dingqi Yang, Hailong Sun, and Philip S Yu. Harnessing multiple large language models: A survey on llm ensemble. *arXiv preprint arXiv:2502.18036*, 2025.

[56] Yu Cheng, Ilias Diakonikolas, Rong Ge, and David P Woodruff. Faster algorithms for high-dimensional robust covariance estimation. In *Conference on Learning Theory*, pages 727–757. PMLR, 2019.

[57] Nadezhda Chirkova, Germán Kruszewski, Jos Rozen, and Marc Dymetman. Should you marginalize over possible tokenizations? In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–12, 2023.

[58] Pavel Chizhov, Catherine Arnett, Elizaveta Korotkova, and Ivan P. Yamshchikov. BPE gets picky: Efficient vocabulary refinement during tokenizer training. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16587–16604, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.925. URL `https://aclanthology.org/2024.emnlp-main.925`.

[59] Christopher A Choquette-Choo, Florian Tramer, Nicholas Carlini, and Nicolas Papernot. Label-only membership inference attacks. In *International conference on machine learning*, pages 1964–1974. PMLR, 2021.

[60] Edward Chou, Florian Tramèr, Giancarlo Pellegrino, and Dan Boneh. Sentinet: Detecting physical attacks against deep learning systems. *arXiv preprint arXiv:1812.00292*, 2018.

[61] Edward Chou, Florian Tramer, and Giancarlo Pellegrino. Sentinet: Detecting localized universal attacks against deep learning systems. In *2020 IEEE Security and Privacy Workshops (SPW)*, pages 48–54. IEEE, 2020.

[62] Yung-Sung Chuang, Yujia Xie, Hongyin Luo, Yoon Kim, James R. Glass, and Pengcheng He. Dola: Decoding by contrasting layers improves factuality in large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

[63] Jonathan H. Clark, Dan Garrette, Iulia Turc, and John Wieting. Canine: Pre-training an efficient tokenization-free encoder for language representation. *Transactions of the Association for Computational Linguistics*, 10:73–91, 2022. doi: 10.1162/tacl_a_00448. URL `https://aclanthology.org/2022.tacl-1.5`.

[64] Together Computer. Redpajama: An open source recipe to reproduce llama training dataset, 2023. URL `https://github.com/togethercomputer/RedPajama-Data`.

[65] Gautier Dagan, Gabriel Synnaeve, and Baptiste Rozière. Getting the most out of your tokenizer for pre-training and domain adaptation. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

[66] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960. URL `https://www.jstor.org/stable/167547`.

[67] Ilias Diakonikolas, Gautam Kamath, Daniel M Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Being robust (in high dimensions) can be practical. In *International Conference on Machine Learning*, pages 999–1008. PMLR, 2017.

[68] Ilias Diakonikolas, Daniel M Kane, and Alistair Stewart. Statistical query lower bounds for robust estimation of high-dimensional gaussians and gaussian mixtures. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–84. IEEE, 2017.

[69] Ilias Diakonikolas, Gautam Kamath, Daniel Kane, Jerry Li, Ankur Moitra, and Alistair Stewart. Robust estimators in high-dimensions without the computational intractability. *SIAM Journal on Computing*, 48(2):742–864, 2019.

[70] Khoa Doan, Yingjie Lao, and Ping Li. Backdoor attack with imperceptible input and latent modification. *Advances in Neural Information Processing Systems*, 34, 2021.

[71] Khoa Doan, Yingjie Lao, Weijie Zhao, and Ping Li. Lira: Learnable, imperceptible and robust backdoor attacks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11966–11976, 2021.

[72] Konstantin Dobler and Gerard De Melo. Focus: Effective embedding initialization for monolingual specialization of multilingual models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13440–13454, 2023.

[73] Yihe Dong, Samuel B Hopkins, and Jerry Li. Quantum entropy scoring for fast robust mean estimation and improved outlier detection. *arXiv preprint arXiv:1906.11366*, 2019.

[74] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In *International conference on machine learning*, pages 1675–1685. PMLR, 2019.

[75] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2018.

[76] Simon S Du, Kangcheng Hou, Russ R Salakhutdinov, Barnabas Poczos, Ruosong Wang, and Keyulu Xu. Graph neural tangent kernel: Fusing graph neural networks with graph kernels. *Advances in neural information processing systems*, 32, 2019.

[77] Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2368–2378, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1246. URL `https://aclanthology.org/N19-1246`.

[78] Michael Duan, Anshuman Suri, Niloofar Mireshghallah, Sewon Min, Weijia Shi, Luke Zettlemoyer, Yulia Tsvetkov, Yejin Choi, David Evans, and Hannaneh Hajishirzi. Do membership inference attacks work on large language models?, 2024. URL `https://arxiv.org/abs/2402.07841`.

[79] Matthew Finlayson, Xiang Ren, and Swabha Swayamdipta. Logits of api-protected llms leak proprietary information, 2024.

[80] Stanislav Fort, Gintare Karolina Dziugaite, Mansheej Paul, Sepideh Kharaghani, Daniel M Roy, and Surya Ganguli. Deep learning versus kernel learning: an empirical study of loss landscape geometry and the time evolution of the neural tangent kernel. *Advances in Neural Information Processing Systems*, 33:5850–5861, 2020.

[81] Hao Fu, Yao; Peng and Tushar Khot. How does gpt obtain its ability? tracing

emergent abilities of language models to their sources. *Yao Fu's Notion*, Dec 2022. URL https://yaofu.notion.site/How-does-GPT-Obtain-its-Ability-Tracing-Emergent-Abilities-of-Language-Models-to-their-Sources-b9a57ac0fcf74f30a1ab9e3e36fa1dc1.

[82] Samir Yitzhak Gadre, Gabriel Ilharco, Alex Fang, Jonathan Hayase, Georgios Smyrnis, Thao Nguyen, Ryan Marten, Mitchell Wortsman, Dhruba Ghosh, Jieyu Zhang, et al. Datacomp: In search of the next generation of multimodal datasets. *Advances in Neural Information Processing Systems*, 36:27092–27112, 2023.

[83] Philip Gage. A new algorithm for data compression. *The C Users Journal archive*, 12: 23–38, 1994. URL https://api.semanticscholar.org/CorpusID:59804030.

[84] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 619–633, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356930. doi: 10.1145/3243734.3243834. URL https://doi.org/10.1145/3243734.3243834.

[85] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*, 2020.

[86] Yansong Gao, Change Xu, Derui Wang, Shiping Chen, Damith C Ranasinghe, and Surya Nepal. Strip: A defence against trojan attacks on deep neural networks. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 113–125, 2019.

[87] Leonidas Gee, Andrea Zugarini, Leonardo Rigutini, Paolo Torroni, et al. Fast vocabulary transfer for language model compression. In *Proceedings of the 2022 Conference on*

Empirical Methods in Natural Language Processing: Industry Track, pages 409–416. Association for Computational Linguistics (ACL), 2022.

[88] Leonidas Gee, Leonardo Rigutini, Marco Ernandes, and Andrea Zugarini. Multi-word tokenization for sequence compression. In Mingxuan Wang and Imed Zitouni, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 612–621, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-industry.58. URL https://aclanthology.org/2023.emnlp-industry.58.

[89] Renato Lui Geh, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van den Broeck. Where is the signal in tokenization space? *arXiv preprint arXiv:2408.08541*, 2024.

[90] Amnon Geifman, Abhay Yadav, Yoni Kasten, Meirav Galun, David Jacobs, and Basri Ronen. On the similarity between the laplace and neural tangent kernels. *Advances in Neural Information Processing Systems*, 33:1451–1461, 2020.

[91] Jonas Geiping, Alex Stein, Manli Shu, Khalid Saifullah, Yuxin Wen, and Tom Goldstein. Coercing llms to do and reveal (almost) anything, 2024. URL https://arxiv.org/abs/2402.14020.

[92] Ariel Gera, Roni Friedman, Ofir Arviv, Chulaka Gunasekara, Benjamin Sznajder, Noam Slonim, and Eyal Shnarch. The benefits of bad advice: Autocontrastive decoding across model layers. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10406–10420, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.580. URL https://aclanthology.org/2023.acl-long.580.

[93] Shafi Goldwasser, Michael P Kim, Vinod Vaikuntanathan, and Or Zamir. Planting

undetectable backdoors in machine learning models. *arXiv preprint arXiv:2204.06974*, 2022.

[94] Dirk Groeneveld, Iz Beltagy, Evan Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, William Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah Smith, and Hannaneh Hajishirzi. OLMo: Accelerating the science of language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15789–15809, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.841. URL https://aclanthology.org/2024.acl-long.841.

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

[96] Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

[97] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

[98] Junfeng Guo and Cong Liu. Practical poisoning attacks on neural networks. In *European Conference on Computer Vision*, pages 142–158. Springer, 2020.

[99] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL `https://www.gurobi.com`.

[100] Jonathan Hayase and Sewoong Oh. Few-shot backdoor attacks via neural tangent kernels. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL `https://openreview.net/forum?id=a70lGJ-rwy`.

[101] Jonathan Hayase, Weihao Kong, Raghav Somani, and Sewoong Oh. Spectre: defending against backdoor attacks using robust statistics. 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 4129–4139. PMLR, 18–24 Jul 2021. URL `https://proceedings.mlr.press/v139/hayase21a.html`.

[102] Jonathan Hayase, Alisa Liu, Yejin Choi, Sewoong Oh, and Noah Smith. Data mixture inference attack: Bpe tokenizers reveal training data compositions. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 8956–8983. Curran Associates, Inc., 2024. doi: 10.52202/079017-0285. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/10e6dfea9a673bef4a7b1cb9234891bc-Paper-Conference.pdf`.

[103] Jonathan Hayase, Alisa Liu, Noah A Smith, and Sewoong Oh. Sampling from your language model one byte at a time. *arXiv preprint arXiv:2506.14123*, 2025.

[104] 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.

[105] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.

[106] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[107] Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *arXiv preprint arXiv:2004.05439*, 2020.

[108] W Ronny Huang, Jonas Geiping, Liam Fowl, Gavin Taylor, and Tom Goldstein. Metapoison: Practical general-purpose clean-label data poisoning. *Advances in Neural Information Processing Systems*, 33:12080–12091, 2020.

[109] Yichong Huang, Xiaocheng Feng, Baohang Li, Yang Xiang, Hui Wang, Ting Liu, and Bing Qin. Ensemble learning for heterogeneous large language models with deep parallel collaboration. *Advances in Neural Information Processing Systems*, 37:119838–119860, 2024.

[110] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In *Advances in Neural Information Processing Systems*, pages 125–136, 2019.

[111] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[112] Jacob Jackson. Character prefix conditioning, 2025. URL `https://www.cursor.com/blog/cpc`.

[113] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. *Advances in neural information processing systems*, 31, 2018.

[114] Arun Jambulapati, Jerry Li, and Kevin Tian. Robust sub-gaussian principal component analysis and width-independent schatten packing. *arXiv preprint arXiv:2006.06980*, 2020.

[115] Mojan Javaheripi, Sébastien Bubeck, Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, et al. Phi-2: The surprising power of small language models. *Microsoft Research Blog*, 1(3):3, 2023.

[116] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b. *arXiv preprint arXiv:2407.10671*, 2023.

[117] Mandar Joshi, Eunsol Choi, Daniel Weld, and Luke Zettlemoyer. TriviaQA: A large scale distantly supervised challenge dataset for reading comprehension. In Regina Barzilay and Min-Yen Kan, editors, *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1601–1611, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1147. URL `https://aclanthology.org/P17-1147`.

[118] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977*, 2019.

[119] Jungo Kasai, Keisuke Sakaguchi, Ronan Le Bras, Hao Peng, Ximing Lu, Dragomir Radev, Yejin Choi, and Noah A. Smith. Twist decoding: Diverse generators guide each other. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of*

the 2022 Conference on Empirical Methods in Natural Language Processing, pages 4909–4923, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.326. URL https://aclanthology.org/2022.emnlp-main.326.

[120] Michael Kearns and Ming Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, 1993.

[121] Diederick P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[122] Pang Wei Koh, Jacob Steinhardt, and Percy Liang. Stronger data poisoning attacks break data sanitization defenses. *Machine Learning*, 111(1):1–47, 2022.

[123] Soheil Kolouri, Aniruddha Saha, Hamed Pirsiavash, and Heiko Hoffmann. Universal litmus patterns: Revealing backdoor attacks in cnns. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 301–310, 2020.

[124] Weihao Kong, Raghav Somani, Sham Kakade, and Sewoong Oh. Robust meta-learning for mixed linear regression with small batches. *arXiv preprint arXiv:2006.09702*, 2020.

[125] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[126] Taku Kudo. Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75, 2018.

[127] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71, 2018.

[128] Sneha Kudugunta, Isaac Caswell, Biao Zhang, Xavier Garcia, Derrick Xin, Aditya Kusupati, Romi Stella, Ankur Bapna, and Orhan Firat. Madlad-400: A multilingual and document-level large audited dataset. *Advances in Neural Information Processing Systems*, 36:67284–67296, 2023.

[129] Dipesh Kumar and Avijit Thawani. BPE beyond word boundary: How NOT to use multi word expressions in neural machine translation. In Shabnam Tafreshi, João Sedoc, Anna Rogers, Aleksandr Drozd, Anna Rumshisky, and Arjun Akula, editors, *Proceedings of the Third Workshop on Insights from Negative Results in NLP*, pages 172–179, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.insights-1.24. URL https://aclanthology.org/2022.insights-1.24.

[130] Kevin A Lai, Anup B Rao, and Santosh Vempala. Agnostic estimation of mean and covariance. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 665–674. IEEE, 2016.

[131] Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training, 2025. URL https://arxiv.org/abs/2411.15124.

[132] Sander Land and Max Bartolo. Fishing for magikarp: Automatically detecting under-trained tokens in large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11631–11646, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.649. URL https://aclanthology.org/2024.emnlp-main.649.

[133] Pavel Laskov. Practical evasion of a learning-based classifier: A case study. In *2014 IEEE symposium on security and privacy*, pages 197–211. IEEE, 2014.

[134] Jaehoon Lee, Samuel Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. *Advances in Neural Information Processing Systems*, 33:15156–15172, 2020.

[135] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. In *Advances in Neural Information Processing Systems*, pages 7167–7177, 2018.

[136] Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alexandros G. Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-lm: In search of the next generation of training sets for language models, 2024. URL `https://arxiv.org/abs/2406.11794`.

[137] Jerry Li and Guanghao Ye. Robust gaussian covariance estimation in nearly-matrix multiplication time. *arXiv preprint arXiv:2006.13312*, 2020.

[138] Margaret Li, Suchin Gururangan, Tim Dettmers, Mike Lewis, Tim Althoff, Noah A. Smith, and Luke Zettlemoyer. Branch-train-merge: Embarrassingly parallel training of expert language models, 2022. URL `https://arxiv.org/abs/2208.03306`.

[139] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *arXiv preprint arXiv:1909.02742*, 2019.

[140] Shaofeng Li, Minhui Xue, Benjamin Zi Hao Zhao, Haojin Zhu, and Xinpeng Zhang. Invisible backdoor attacks on deep neural networks via steganography and regularization. *IEEE Transactions on Dependable and Secure Computing*, 18(5):2088–2105, 2020.

[141] Xiang Lisa Li, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis. Contrastive decoding: Open-ended text generation as optimization. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12286–12312, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.687. URL `https://aclanthology.org/2023.acl-long.687`.

[142] Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*, 2023.

[143] Zhiyuan Li, Ruosong Wang, Dingli Yu, Simon S Du, Wei Hu, Ruslan Salakhutdinov, and Sanjeev Arora. Enhanced convolutional neural tangent kernels. *arXiv preprint arXiv:1911.00809*, 2019.

[144] Shiyu Liang, Yixuan Li, and Rayadurgam Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. *arXiv preprint arXiv:1706.02690*, 2017.

[145] Shiyu Liang, Yixuan Li, and R Srikant. Enhancing the reliability of out-of-distribution image detection in neural networks. In *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[146] Tomasz Limisiewicz, Terra Blevins, Hila Gonen, Orevaoghene Ahia, and Luke Zettlemoyer. MYTE: Morphology-driven byte encoding for better and fairer multilingual language modeling. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15059–15076, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.804. URL https://aclanthology.org/2024.acl-long.804.

[147] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[148] Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[149] Alisa Liu, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi. DExperts: Decoding-time controlled text generation with experts and anti-experts. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 6691–6706, Online, August 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.acl-long.522. URL https://aclanthology.org/2021.acl-long.522.

[150] Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A Smith. Tuning language models by proxy. In *First Conference on Language Modeling*, 2024.

[151] Alisa Liu, Jonathan Hayase, Valentin Hofmann, Sewoong Oh, Noah A Smith, and Yejin

Choi. SuperBPE: Space travel for language models. *arXiv preprint arXiv:2503.13423*, 2025. URL `https://arxiv.org/abs/2503.13423`.

[152] Cong Liu, Xiaojun Quan, Yan Pan, Liang Lin, Weigang Wu, and Xu Chen. Cool-fusion: Fuse large language models without training. *arXiv preprint arXiv:2407.19807*, 2024.

[153] Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. Fine-pruning: Defending against backdooring attacks on deep neural networks. In *International Symposium on Research in Attacks, Intrusions, and Defenses*, pages 273–294. Springer, 2018.

[154] Xiyang Liu, Weihao Kong, Sham Kakade, and Sewoong Oh. Robust and differentially private mean estimation. *arXiv preprint arXiv:2102.09159*, 2021.

[155] Yihong Liu, Peiqin Lin, Mingyang Wang, and Hinrich Schütze. Ofa: A framework of initializing unseen subword embeddings for efficient large-scale multilingual continued pretraining. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1067–1097, 2024.

[156] Yingqi Liu, Shiqing Ma, Yousra Aafer, Wen-Chuan Lee, Juan Zhai, Weihang Wang, and Xiangyu Zhang. Trojaning attack on neural networks. 2017.

[157] Yunfei Liu, Xingjun Ma, James Bailey, and Feng Lu. Reflection backdoor: A natural backdoor attack on deep neural networks. In *European Conference on Computer Vision*, pages 182–199. Springer, 2020.

[158] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.

[159] Philip M Long. Properties of the after kernel. *arXiv preprint arXiv:2105.10585*, 2021.

[160] Shayne Longpre, Gregory Yauney, Emily Reif, Katherine Lee, Adam Roberts, Barret Zoph, Denny Zhou, Jason Wei, Kevin Robinson, David Mimno, and Daphne Ippolito. A pretrainer's guide to training data: Measuring the effects of data age, domain

coverage, quality, & toxicity. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 3245–3276, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.179. URL https://aclanthology.org/2024.naacl-long.179.

[161] Scott Lundberg. The art of prompt design: Prompt boundaries and token healing, 2023. URL https://medium.com/towards-data-science/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38.

[162] Bo Lv, Chen Tang, Yanan Zhang, Xin Liu, Yue Yu, and Ping Luo. Specfuse: Ensembling large language models via next-segment prediction. *arXiv preprint arXiv:2412.07380*, 2024.

[163] YINGWEI MA, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. At which training stage does code data help LLMs reasoning? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KIPJKST4gw.

[164] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[165] Kelly Marchisio, Patrick Lewis, Yihong Chen, and Mikel Artetxe. Mini-model adaptation: Efficiently extending pretrained models to new languages via aligned shallow training. In *The 61st Annual Meeting Of The Association For Computational Linguistics*, 2023.

[166] Justus Mattern, Fatemehsadat Mireshghallah, Zhijing Jin, Bernhard Schoelkopf, Mrinmaya Sachan, and Taylor Berg-Kirkpatrick. Membership inference attacks against

language models via neighbourhood comparison. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 11330–11343, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.719. URL https://aclanthology.org/2023.findings-acl.719.

[167] Costas Mavromatis, Petros Karypis, and George Karypis. Pack of llms: Model fusion at test-time via perplexity optimization. In *First Conference on Language Modeling*, 2024.

[168] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. Kernel methods through the roof: handling billions of points efficiently. *Advances in Neural Information Processing Systems*, 33:14410–14422, 2020.

[169] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 691–706, 2019. doi: 10.1109/SP.2019.00029. URL https://ieeexplore.ieee.org/document/8835269.

[170] Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[171] Sabrina J. Mielke. Can you compare perplexity across different segmentations?, Apr 2019. URL https://sjmielke.com/comparing-perplexities.htm.

[172] Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. Wechsel: Effective initialization of subword embeddings for cross-lingual transfer of monolingual language models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3992–4006, 2022.

[173] Benjamin Minixhofer, Edoardo Ponti, and Ivan Vulić. Zero-shot tokenizer transfer. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[174] Benjamin Minixhofer, Ivan Vulić, and Edoardo Maria Ponti. Universal cross-tokenizer distillation via approximate likelihood matching. *arXiv preprint arXiv:2503.20083*, 2025.

[175] Fatemehsadat Mireshghallah, Kartik Goyal, Archit Uniyal, Taylor Berg-Kirkpatrick, and Reza Shokri. Quantifying privacy risks of masked language models using membership inference attacks. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8332–8347, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.570. URL `https://aclanthology.org/2022.emnlp-main.570`.

[176] Niloofar Mireshghallah, Nikolai Vogler, Junxian He, Omar Florez, Ahmed El-Kishky, and Taylor Berg-Kirkpatrick. Simple temporal adaptation to changing label sets: Hashtag prediction via dense KNN. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7302–7311, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.452. URL `https://aclanthology.org/2023.emnlp-main.452`.

[177] Mehran Mozaffari-Kermani, Susmita Sur-Kolay, Anand Raghunathan, and Niraj K Jha. Systematic poisoning attacks on and defenses for machine learning in healthcare. *IEEE journal of biomedical and health informatics*, 19(6):1893–1905, 2014.

[178] Luis Muñoz-González, Battista Biggio, Ambra Demontis, Andrea Paudice, Vasin Wongrassamee, Emil C Lupu, and Fabio Roli. Towards poisoning of deep learning algorithms with back-gradient optimization. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 27–38, 2017.

[179] Milad Nasr, Nicholas Carlini, Jonathan Hayase, Matthew Jagielski, A. Feder Cooper, Daphne Ippolito, Christopher A. Choquette-Choo, Eric Wallace, Florian Tramèr, and

Katherine Lee. Scalable extraction of training data from (production) language models, 2023. URL https://arxiv.org/abs/2311.17035.

[180] Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. Efficient transformers with dynamic token pooling. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6403–6417, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.353. URL https://aclanthology.org/2023.acl-long.353.

[181] James Newsome, Brad Karp, and Dawn Song. Paragraph: Thwarting signature learning by training maliciously. In *International Workshop on Recent Advances in Intrusion Detection*, pages 81–105. Springer, 2006.

[182] Timothy Nguyen, Zhourong Chen, and Jaehoon Lee. Dataset meta-learning from kernel ridge-regression. In *International Conference on Learning Representations*, 2020.

[183] Timothy Nguyen, Roman Novak, Lechao Xiao, and Jaehoon Lee. Dataset distillation with infinitely wide convolutional networks. *Advances in Neural Information Processing Systems*, 34, 2021.

[184] Tuan Anh Nguyen and Anh Tuan Tran. Wanet-imperceptible warping-based backdoor attack. In *International Conference on Learning Representations*, 2020.

[185] Roman Novak, Lechao Xiao, Jiri Hron, Jaehoon Lee, Alexander A. Alemi, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. Neural tangents: Fast and easy infinite neural networks in python. In *International Conference on Learning Representations*, 2020. URL https://github.com/google/neural-tangents.

[186] Roman Novak, Jascha Sohl-Dickstein, and Samuel Stern Schoenholz. Fast finite width neural tangent kernel. In *Fourth Symposium on Advances in Approximate Bayesian Inference*, 2021.

[187] Byung-Doh Oh and William Schuler. Leading whitespaces of language models' subword vocabulary pose a confound for calculating word probabilities. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3464–3472, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.202. URL https://aclanthology.org/2024.emnlp-main.202.

[188] Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Michal Guerquin, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James V. Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 olmo 2 furious, 2024. URL https://arxiv.org/abs/2501.00656.

[189] OpenAI. Introducing ChatGPT, 2022. URL https://openai.com/index/chatgpt/.

[190] OpenAI. Openai platform documentation, 2023. URL https://platform.openai.com/docs. Accessed: 2025/05/10.

[191] OpenAI. Hello GPT-4o, 2024. URL https://openai.com/index/hello-gpt-4o/.

[192] Artidoro Pagnoni, Ram Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srinivasan Iyer. Byte latent transformer: Patches scale better than tokens, 2024. URL https://arxiv.org/abs/2412.09871.

[193] Denis Paperno, Germán Kruszewski, Angeliki Lazaridou, Ngoc Quan Pham, Raffaella Bernardi, Sandro Pezzelle, Marco Baroni, Gemma Boleda, and Raquel Fernández.

The LAMBADA dataset: Word prediction requiring a broad discourse context. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

[194] Lana Periša. *Recompression of Hadamard products of tensors in Tucker format*. PhD thesis, University of Zagreb. Faculty of Science. Department of Mathematics, 2017.

[195] Buu Phan, Marton Havasi, Matthew Muckley, and Karen Ullrich. Understanding and mitigating tokenization bias in language models, 2024. URL https://arxiv.org/abs/2406.16829.

[196] Buu Phan, Brandon Amos, Itai Gat, Marton Havasi, Matthew J Muckley, and Karen Ullrich. Exact byte-level probabilities from tokenized language models for fim-tasks and model ensembles. In *The Thirteenth International Conference on Learning Representations*, 2025.

[197] Krishna Pillutla, Sham M Kakade, and Zaid Harchaoui. Robust aggregation for federated learning. *arXiv preprint arXiv:1912.13445*, 2019.

[198] Tiago Pimentel and Clara Meister. How to compute the probability of a word. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 18358–18375, 2024.

[199] Ivan Provilkov, Dmitrii Emelianenko, and Elena Voita. BPE-dropout: Simple and effective subword regularization. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1882–1892, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.170. URL https://aclanthology.org/2020.acl-main.170.

[200] Erwin Quiring and Konrad Rieck. Backdooring and poisoning neural networks with image-scaling attacks. *arXiv preprint arXiv:2003.08633*, 2020.

[201] Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019. URL `https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

[202] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. SQuAD: 100,000+ questions for machine comprehension of text. In Jian Su, Kevin Duh, and Xavier Carreras, editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL `https://aclanthology.org/D16-1264`.

[203] Ambrish Rawat, Killian Levacher, and Mathieu Sinn. The devil is in the gan: Defending deep generative models against backdoor attacks. *arXiv preprint arXiv:2108.01644*, 2021.

[204] Marco Tulio Ribeiro. A guidance language for controlling large language models, 2023. URL `https://github.com/guidance-ai/guidance?tab=readme-ov-file#text-not-tokens`.

[205] Benjamin IP Rubinstein, Blaine Nelson, Ling Huang, Anthony D Joseph, Shing-hon Lau, Satish Rao, Nina Taft, and J Doug Tygar. Antidote: understanding and defending against poisoning of anomaly detectors. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 1–14, 2009.

[206] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. Falkon: An optimal large scale kernel method. *Advances in neural information processing systems*, 30, 2017.

[207] Jessica Rumbelow and Matthew Watkins. Solidgoldmagikarp (plus, prompt generation), 2023. URL https://www.lesswrong.com/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation.

[208] Ruslan S. 4d masks support in transformers, 2024. URL https://huggingface.co/blog/poedator/4d-masks.

[209] Aniruddha Saha, Akshayvarun Subramanya, and Hamed Pirsiavash. Hidden trigger backdoor attacks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 11957–11965, 2020.

[210] Ahmed Salem, Yannick Sautter, Michael Backes, Mathias Humbert, and Yang Zhang. Baaan: Backdoor attacks against autoencoder and gan-based machine learning models. *arXiv preprint arXiv:2010.03007*, 2020.

[211] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, et al. Laion-5b: An open large-scale dataset for training next generation image-text models. *Advances in neural information processing systems*, 35:25278–25294, 2022.

[212] Mike Schuster and Kaisuke Nakajima. Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE, 2012.

[213] Anton Schäfer, Shauli Ravfogel, Thomas Hofmann, Tiago Pimentel, and Imanol Schlag. Language imbalance can boost cross-lingual generalisation, 2024. URL https://arxiv.org/abs/2404.07982.

[214] Mariia Seleznova and Gitta Kutyniok. Analyzing finite neural networks: Can we trust neural tangent kernel theory? In *Mathematical and Scientific Machine Learning*, pages 868–895. PMLR, 2022.

[215] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. In Katrin Erk and Noah A. Smith, editors, *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1162. URL https://aclanthology.org/P16-1162.

[216] 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.

[217] Ruizhe Shi, Yifang Chen, Yushi Hu, Alisa Liu, Hannaneh Hajishirzi, Noah A. Smith, and Simon Shaolei Du. Decoding-time language model alignment with multiple objectives. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[218] Weijia Shi, Anirudh Ajith, Mengzhou Xia, Yangsibo Huang, Daogao Liu, Terra Blevins, Danqi Chen, and Luke Zettlemoyer. Detecting pretraining data from large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=zWqr3MQuNs.

[219] Weijia Shi, Xiaochuang Han, Mike Lewis, Yulia Tsvetkov, Luke Zettlemoyer, and Wen-tau Yih. Trusting your evidence: Hallucinate less with context-aware decoding. In Kevin Duh, Helena Gomez, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 2: Short Papers)*, pages 783–791, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-short.69. URL https://aclanthology.org/2024.naacl-short.69.

[220] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.

[221] Reza Shokri et al. Bypassing backdoor detection algorithms in deep learning. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 175–183. IEEE, 2020.

[222] Noah A. Smith and Ryan Tromble. Sampling uniformly from the unit simplex. Technical report, 2004. URL https://www.cs.cmu.edu/~nasmith/papers/smith+tromble.tr04.pdf.

[223] Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. Certified defenses for data poisoning attacks. In *Advances in neural information processing systems*, pages 3517–3529, 2017.

[224] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H Brendan McMahan. Can you really backdoor federated learning? *arXiv preprint arXiv:1911.07963*, 2019.

[225] Anshuman Suri and David Evans. Formalizing and estimating distribution inference risks. In *Privacy Enhancing Technologies Symposium*, volume 2022, pages 528–551, 2022. URL https://petsymposium.org/popets/2022/popets-2022-0121.php.

[226] Andrei-Valentin Tănase and Elena Pelican. Supratok: Cross-boundary tokenization for enhanced language model performance. *arXiv preprint arXiv:2508.11857*, 2025.

[227] Yi Tay, Vinh Q. Tran, Sebastian Ruder, Jai Gupta, Hyung Won Chung, Dara Bahri, Zhen Qin, Simon Baumgartner, Cong Yu, and Donald Metzler. Charformer: Fast character transformers via gradient-based subword tokenization. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=JtBRnrlOEFN.

[228] Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*, 2024.

[229] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[230] Gemma Team, Aishwarya Kamath, Johan Ferret, Shreya Pathak, Nino Vieillard, Ramona Merhej, Sarah Perrin, Tatiana Matejovicova, Alexandre Ramé, Morgane Rivière, et al. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

[231] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[232] Llama Team. Introducing llama 3.1: Our most capable models to date, 2024. URL `https://ai.meta.com/blog/meta-llama-3-1/`.

[233] Llama Team. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models, 2024. URL `https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/`. Accessed: 2025/05/10.

[234] Llama Team. The llama 4 herd: The beginning of a new era of natively multimodal ai innovation, Apr 2025. URL `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`.

[235] Mistral AI Team. Mistral NeMo | Mistral AI — mistral.ai. `https://mistral.ai/news/mistral-nemo`, 2024. [Accessed 25-09-2025].

[236] Olmo Team. Olmo release notes, 2025. URL `https://allenai.org/olmo/release-notes#olmo-2-1b`. Accessed: 2025/05/10.

[237] Qwen Team. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2, 2024.

[238] Qwen Team. Qwen3: Think deeper, act faster, 2025. URL https://qwenlm.github.io/blog/qwen3/. Accessed: 2025/05/10.

[239] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[240] 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.

[241] Brandon Tran, Jerry Li, and Aleksander Madry. Spectral signatures in backdoor attacks. *Advances in neural information processing systems*, 31, 2018.

[242] Ke Tran. From english to foreign languages: Transferring pre-trained language models. *arXiv preprint arXiv:2002.07306*, 2020.

[243] Bojan Tunguz. 200,000+ jeopardy! questions, 2019. URL https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions.

[244] Bojan Tunguz. 200,000+ jeopardy! questions, 2019. URL https://www.kaggle.com/datasets/tunguz/200000-jeopardy-questions.

[245] Anil Turaga. Character prefix conditioning with back tokenization, 2025. URL https://anilturaga.github.io/cpc.

[246] Alexander Turner, Dimitris Tsipras, and Aleksander Madry. Label-consistent backdoor attacks. *arXiv preprint arXiv:1912.02771*, 2019.

[247] Hendrik van Antwerpen and Alexander Neubeck. So many tokens, so little time: Introducing a faster, more flexible byte-pair tokenizer, 2025. URL https://github

.blog/ai-and-ml/llms/so-many-tokens-so-little-time-introducing-a-faster-more-flexible-byte-pair-tokenizer/. Accessed: 2025/05/10.

[248] Tim Vieira, Ben LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J O'Donnell, and Ryan Cotterell. From language models over tokens to language models over characters. *arXiv preprint arXiv:2412.03719*, 2024.

[249] Tim Vieira, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian DuSell, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J O'Donnell, and Ryan Cotterell. Language models over canonical byte-pair encodings. *arXiv preprint arXiv:2506.07956*, 2025.

[250] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[251] Andrea Walther and Andreas Griewank. Advantages of binomial checkpointing for memory-reduced adjoint calculations. In *Numerical mathematics and advanced applications*, pages 834–843. Springer, 2004.

[252] Binghui Wang, Xiaoyu Cao, Neil Zhenqiang Gong, et al. On certifying robustness against backdoor attacks via randomized smoothing. *arXiv preprint arXiv:2002.11750*, 2020.

[253] Bolun Wang, Yuanshun Yao, Shawn Shan, Huiying Li, Bimal Viswanath, Haitao Zheng,

and Ben Y Zhao. Neural cleanse: Identifying and mitigating backdoor attacks in neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 707–723. IEEE, 2019.

[254] Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, Harit Vishwakarma, Saurabh Agarwal, Jy-yong Sohn, Kangwook Lee, and Dimitris Papailiopoulos. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[255] Junxiong Wang, Tushaar Gangavarapu, Jing Nathan Yan, and Alexander M Rush. Mambabyte: Token-free selective state space model. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=X1xNsuKssb`.

[256] Maurice Weber, Xiaojun Xu, Bojan Karlaš, Ce Zhang, and Bo Li. Rab: Provable robustness against backdoor attacks. *arXiv preprint arXiv:2003.08904*, 2020.

[257] BigScience Workshop. Bloom: A 176b-parameter open-access multilingual language model, 2023. URL `https://arxiv.org/abs/2211.05100`.

[258] Pengfei Xia, Hongjing Niu, Ziqiang Li, and Bin Li. Enhancing backdoor attacks with multi-level mmd regularization. *IEEE Transactions on Dependable and Secure Computing*, 2022.

[259] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698, 2015.

[260] Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. Dba: Distributed backdoor attacks against federated learning. In *International Conference on Learning Representations*, 2019.

[261] Yiqing Xie, Atharva Naik, Daniel Fried, and Carolyn Rose. Data augmentation for code translation with comparable corpora and multiple references. In Houda Bouamor,

Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 13725–13739, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-emnlp.917. URL https://aclanthology.org/2023.findings-emnlp.917.

[262] Hao Xu, Alisa Liu, Jonathan Hayase, Yejin Choi, and Noah A. Smith. Are you going to finish that? a practical study of the partial token problem, 2026. URL https://arxiv.org/abs/2601.23223.

[263] Yangyifan Xu, Jianghao Chen, Junhong Wu, and Jiajun Zhang. Hit the sweet spot! span-level ensemble for large language models. *arXiv preprint arXiv:2409.18583*, 2024.

[264] Yangyifan Xu, Jinliang Lu, and Jiajun Zhang. Bridging the gap between different vocabularies for llm ensemble. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 7133–7145, 2024.

[265] Yangyifan Xu, Jianghao Chen, Junhong Wu, and Jiajun Zhang. Hit the sweet spot! span-level ensemble for large language models. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 8314–8325, 2025.

[266] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

[267] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[268] Chaofei Yang, Qing Wu, Hai Li, and Yiran Chen. Generative poisoning attack method against neural networks. *arXiv preprint arXiv:1703.01340*, 2017.

[269] Greg Yang. Tensor programs ii: Neural tangent kernel for any architecture. *arXiv preprint arXiv:2006.14548*, 2020.

[270] Jinbiao Yang. Rethinking tokenization: Crafting better tokenizers for large language models, 2024. URL `https://arxiv.org/abs/2403.00417`.

[271] Yuanshun Yao, Huiying Li, Haitao Zheng, and Ben Y Zhao. Latent backdoor attacks on deep neural networks. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2041–2055, 2019.

[272] Alex Young, Bei Chen, Chao Li, Chengen Huang, Ge Zhang, Guanwei Zhang, Guoyin Wang, Heng Li, Jiangcheng Zhu, Jianqun Chen, et al. Yi: Open foundation models by 01. ai. *arXiv preprint arXiv:2403.04652*, 2024.

[273] Lili Yu, Daniel Simig, Colin Flaherty, Armen Aghajanyan, Luke Zettlemoyer, and Mike Lewis. MEGABYTE: Predicting million-byte sequences with multiscale transformers. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL `https://openreview.net/forum?id=JTmO2V9Xpz`.

[274] Yao-Ching Yu, Chun Chih Kuo, Ye Ziqi, Chang Yucheng, and Yueh-Se Li. Breaking the ceiling of the llm community by treating token generation as a classification for ensembling. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 1826–1839, 2024.

[275] Chia-Hung Yuan and Shan-Hung Wu. Neural tangent generalization attacks. In *International Conference on Machine Learning*, pages 12230–12240. PMLR, 2021.

[276] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *British Machine Vision Conference 2016*. British Machine Vision Association, 2016.

[277] Amir Zandieh, Insu Han, Haim Avron, Neta Shoham, Chaewon Kim, and Jinwoo Shin. Scaling neural tangent kernels via sketching and random features. *Advances in Neural Information Processing Systems*, 34, 2021.

[278] Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. *arXiv preprint arXiv:2508.06471*, 2025.

[279] Vivienne Zhang, Shashank Verma, Neal Vaidya, Abhishek Sawarkar, and Amanda Saunders. Nvidia ai foundation models: Build custom enterprise chatbots and co-pilots with production-ready llms, Nov 2024. URL `https://developer.nvidia.com/blog/nvidia-ai-foundation-models-build-custom-enterprise-chatbots-and-co-pilots-with-production-ready-llms/`.

[280] Wanrong Zhang, Shruti Tople, and Olga Ohrimenko. Leakage of dataset properties in Multi-Party machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2687–2704. USENIX Association, August 2021. ISBN 978-1-939133-24-3. URL `https://www.usenix.org/conference/usenixsecurity21/presentation/zhang-wanrong`.

[281] Shihao Zhao, Xingjun Ma, Xiang Zheng, James Bailey, Jingjing Chen, and Yu-Gang Jiang. Clean-label backdoor attacks on video recognition models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14443–14452, 2020.

[282] Wenhao Zheng, Yixiao Chen, Weitong Zhang, Souvik Kundu, Yun Li, Zhengzhong Liu, Eric P Xing, Hongyi Wang, and Huaxiu Yao. Citer: Collaborative inference for efficient large language model decoding with token-level routing. *arXiv preprint arXiv:2502.01976*, 2025.

[283] Haoti Zhong, Cong Liao, Anna Cinzia Squicciarini, Sencun Zhu, and David Miller.

Backdoor embedding in convolutional neural network models via invisible perturbation. In *Proceedings of the Tenth ACM Conference on Data and Application Security and Privacy*, pages 97–108, 2020.

[284] Junhao Zhou, Yufei Chen, Chao Shen, and Yang Zhang. Property inference attacks against gans. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022*. The Internet Society, 2022. URL `https://www.ndss-symposium.org/ndss-paper/auto-draft-240/`.

[285] Yufan Zhou, Zhenyi Wang, Jiayi Xian, Changyou Chen, and Jinhui Xu. Meta-learning with neural tangent kernels. *arXiv preprint arXiv:2102.03909*, 2021.

[286] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on mathematical software (TOMS)*, 23(4):550–560, 1997.

[287] Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.