Anytime-Valid Detection of Unauthorized Data Use in Authentication and Machine Learning Systems

by

Zonghao Huang

Department of Computer Science
Duke University

Defense Date: March 31, 2026

Approved:

Michael K. Reiter, Advisor

Lujo Bauer

Bhuwan Dhingra

Neil Zhenqiang Gong

Dissertation submitted in partial fulfillment of the requirements for the degree of
Doctor of Philosophy in the Department of Computer Science
in the Graduate School of Duke University
2026

ABSTRACT

Anytime-Valid Detection of Unauthorized Data Use in Authentication and Machine Learning Systems

by

Zonghao Huang

Department of Computer Science
Duke University

Defense Date: March 31, 2026

Approved:

Michael K. Reiter, Advisor

Lujo Bauer

Bhuwan Dhingra

Neil Zhenqiang Gong

An abstract of a dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science in the Graduate School of Duke University
2026

Abstract

Unauthorized data use—whether through credential-database compromise or the incorporation of datasets into machine learning (ML) models' training without data owners' consent—poses a persistent threat to security, privacy, and data sovereignty. Detecting such unauthorized data use in computing systems requires mechanisms that remain effective and reliable even against adaptive adversaries. This dissertation develops anytime-valid detection frameworks for identifying unauthorized data use in authentication and machine learning systems, which provide tunable and provably bounded false-detection guarantees that remain valid under continuous evidence accumulation and adaptive stopping.

The first part of this dissertation studies credential-database breach detection using honeywords (also known as decoy passwords). We begin by analyzing the security of existing honeyword-based methods under realistic threat models where breach attackers exploit cross-site password knowledge and false-alarm attackers attempt to trigger breach alarms without breaching the credential database. Our analysis shows that state-of-the-art schemes fail to achieve near-ideal false-negative rates and are prone to false alarms, limiting their practical deployment. Building on these insights, we introduce LeakSentinel, a honeyword framework for anytime-valid detection of credential-database breaches. LeakSentinel guarantees a tunable and provably bounded global false-detection rate, meaning that in the absence of a breach, the probability of ever raising an alarm over an infinite sequence of login attempts is tunable and provably bounded. At the same time, it maintains strong detection power against adaptive breach attack strategies.

The second part addresses unauthorized data use in ML systems. We propose a general data-use auditing framework that enables a data owner to test whether her dataset has been used in model training. The framework achieves anytime-valid detection in that the data owner may continuously accumulate data-use evidence through querying the audited ML model and adaptively stop at any time while maintaining a tunable, provably bounded false-detection rate. Finally, we explore the limits of data-use auditing in adversarial scenarios through AcidWash, a framework for purifying auditable training data to mitigate data-use

auditing in ML models. AcidWash demonstrates how adversaries can mitigate detection while preserving model utility, providing a useful tool to evaluate the robustness of data-use auditing methods.

Together, these contributions establish a principled foundation for anytime-valid detection of unauthorized data use and clarify the capabilities of statistical auditing in security-critical systems.

Dedication

To my parents and my grandma.

Contents

List of Tables

List of Figures

Acknowledgements

This dissertation would not have been possible without the support, guidance, and encouragement of many people.

First, I owe my deepest thanks to my advisor, Prof. Michael Reiter, for taking me on as his student and guiding me throughout my Ph.D. journey. His insight, encouragement, and unwavering support have profoundly shaped me as a researcher. I am deeply grateful for his thoughtful advice, patience, and for continually encourage me to stay persistent when facing research challenges.

I am also sincerely grateful to my collaborator, Prof. Neil Gong, whose expertise in machine learning security greatly influenced my research. His guidance and discussions were invaluable to my work on data-use auditing. I thank Prof. Lujo Bauer for sharing his expertise in password security and for his valuable guidance on my honeyword analysis project. I am equally grateful to my committee member Prof. Bhuwan Dhingra for insightful discussions on large language models and machine learning, which broadened my perspective in ways that meaningfully strengthened my research.

I would also like to thank my internship mentors, Dr. Coby Wang and Dr. Sunpreet Arora, for their mentorship and support during my internships at VISA Research.

I am fortunate to have many wonderful friends who kept me grounded throughout this journey. I thank my friends at Duke for their friendship and encouragement.

Finally, I owe my deepest gratitude to my family. I thank my parents for their unconditional love and sacrifices, and my grandmother for her constant care and support. Their belief in me has always been my greatest source of strength.

1. Introduction

From cloud services and online platforms to financial systems, healthcare infrastructures, e-commerce platforms, Internet-of-Things (IoT) ecosystems, and AI-driven applications, data has become a foundational operational asset. Modern computing systems continuously collect, store, and process vast amounts of data, including user credentials, personal information, behavioral logs, transaction records, telemetry streams, and proprietary datasets. These data assets power service personalization, fraud detection, access control, decision-making, automation, and artificial intelligence at scale. For example, access-control mechanisms depend on stored authentication data to enforce security boundaries. Recommendation engines rely on behavioral logs to personalize content. Financial platforms analyze transaction histories to detect fraud. Healthcare systems process medical records to support diagnosis and treatment. Machine learning systems leverage curated datasets to train predictive and generative models. Across these diverse domains, the integrity and authorized use of data are central to system correctness and trustworthiness.

As a result, unauthorized data use has emerged as a critical cross-cutting challenge in modern computing. Broadly speaking, unauthorized data use refers to any access, replication, or utilization of data beyond the scope intended or permitted by its owner. Such misuse may occur due to external attacks (e.g., database breaches), insider threats, supply-chain compromise, or unauthorized collection for model training. The consequences range from privacy violations and identity theft to intellectual property disputes, regulatory penalties, and erosion of user trust. Increasing legal scrutiny and regulatory frameworks further underscore the need for principled mechanisms that can detect whether data has been used without the data owner's permission.

In this dissertation, we focus on two fundamental and representative forms of unauthorized data use:

- Credential-database breaches in authentication systems, where attackers breach the credential database to steal stored authentication data and attempt to exploit it;
- Unauthorized use of data in model training of ML systems, where proprietary or

personal datasets are incorporated into training ML models without the data owner's consent.

These two problems capture distinct yet structurally related manifestations of unauthorized data use. In authentication systems, the challenge is to detect whether a credential database has been compromised and subsequently exploited through login behavior. Such detection mechanisms must provide provably bounded false-detection rates, since false breach alarms are disruptive, costly, and themselves exploitable by adversaries. In ML systems, the challenge is to determine whether a model has been trained on a particular dataset or data instance without authorization of data owners. Here, detection mechanisms must likewise offer rigorous and tunable false-detection guarantees so that their conclusions remain reliable and defensible in high-stakes settings.

Despite their differences, both problems reduce to a common question: how can a data owner continuously accumulate evidence of data use and adaptively stop at any time while maintaining strict control over false detections? Addressing this question requires *anytime-valid detection*, where the probability of ever triggering a false detection remains provably bounded regardless of when the data owner stops accumulating data-use evidence. Designing detection frameworks with such guarantees under adversarial and adaptive conditions forms the central theme of this dissertation.

1.1 Detecting Credential-Database Breaches

Credential-database breaches: Credential-database breaches have long been a pervasive security problem and continue to grow in scale and frequency. Such breaches are the largest source of exposed passwords on the Internet. In mid-2025, reports from cybersecurity researchers revealed that an unprecedented 16 billion credentials associated with major social media platforms, including Apple, Google, Meta, and Telegram, had been exposed online [31]. These leaked credentials provide cybercriminals with what has been described as "a blueprint for mass exploitation" [31]. They enable massive account takeovers, in which attackers hijack social media, financial, and corporate accounts using stolen cre-

dentials [125], resulting in billions of dollars in aggregate losses [92]. Moreover, exposed credentials fuel credential-stuffing attacks [144], where attackers systematically reuse leaked username-password pairs across multiple services, due to the tendency of users to reuse passwords across sites [110, 164]. The impact of credential-database breaches is exacerbated by delayed detection. According to IBM Cost of a Data Breach Report 2025 [63, p. 12], the average time to identify a data breach is 241 days. This prolonged window of vulnerability gives attackers ample opportunity to crack passwords offline, monetize stolen credentials through underground markets, or directly exploit compromised accounts before any breach response is triggered [125, 144]. As a result, even a single undetected credential-database breach can cause sustained and widespread harm to both users and service providers.

Detecting credential-database breaches using honeywords: A methodology for accelerating credential-database breach detection was proposed by Juels and Rivest in 2013 [69], which augments a site's credential database with decoy passwords, known as honeywords. Each user password is stored alongside multiple honeywords, thereby hiding the user password among honeywords. If an attacker breaches the credential database and subsequently attempts to exploit the stolen credentials, he may inadvertently submit honeywords in login attempts, which might alert the site to its breach. Honeyword-based detection relies on two central requirements. First, an attacker who has compromised the credential database should be unable to reliably distinguish each user password from its associated honeywords. This property is commonly quantified by flatness [69], which measures the degree to which each account honeyword appears to the attacker to be at least as plausible as the user password. Second, it requires that it should be difficult for any party that has not breached the credential database (e.g., false-alarm attackers or legitimate users) to trigger breach alarms. This requirement is particularly critical in practice because breach responses are costly and disruptive for triggering investigations and response procedures. According to IBM Cost of a Data Breach Report 2025 [63, p. 13], the average cost for breach detection and escalation is $1.47 million, covering forensic and investigative activities, assessment and audit services, crisis management, and communications to executives and boards. In addition,

unreliable detection that is prone to false alarm can undermine operational trust in the defense mechanism.

Research gaps in detecting credential-database breaches: The widespread tendency of users to reuse passwords across sites (e.g., [110, 164]) fundamentally challenges honeyword-based breach detection. If an attacker obtains a user's passwords from other compromised services through database breaches, phishing, or malware, the attacker can leverage this cross-site knowledge to distinguish the user password from its associated honeywords. Although this threat model may appear stringent, it is increasingly realistic. A mid-2025 report identified over 16 billion credentials circulating in cybercriminal marketplaces [31], averaging more than two credentials per person worldwide. Under such conditions, it remains unclear whether existing honeyword schemes can achieve near-ideal detection performance. This motivates a systematic re-examination of honeyword-based detection under realistic cross-site threat models.

In addition, achieving both good flatness and a small false-detection rate in a honeyword method has proven elusive [160]. Existing works [4, 44, 52, 157, 175] primarily focus on improving flatness by generating honeywords that are difficult to distinguish from user passwords. However, these approaches typically rely on a simple detection mechanism that raises a breach alarm whenever a sufficient number of honeywords are observed in login attempts. As a result, they suffer from a high and unbounded false-detection rate in the presence of false-alarm attackers, which significantly limits the practical applications of these honeyword-based methods. Consequently, an important and largely unresolved problem is to design a honeyword-based breach detection framework that remains effective under realistic breach attacks while providing tunable and provably bounded global false-detection rates, even in the presence of false-alarm attacks.

1.2 Auditing Data Use in Training ML Models

Concerns on unauthorized data use in ML: The rapid advances of machine learning (ML) depend on the availability of massive amounts of training data. However, the developers of

these ML models often do not disclose the origins of their training data, raising legal disputes over unauthorized data use in training ML models. For example, in 2020, multiple lawsuits were filed against Clearview AI, claiming that Clearview AI scraped millions of photos online to train its facial recognition models, violating the rights of the users of those images [56]. In 2023, The New York Times filed a lawsuit against OpenAI and Microsoft, claiming that millions of articles published by The New York Times were used without authorization by OpenAI and Microsoft to train their ML models [143]. Recently, California passed AI legislation [1] that underscores the importance of tracing the origins of data used in training ML models. In addition, established data regulations, such as the General Data Protection Regulation (GDPR) in Europe [94], the California Consumer Privacy Act (CCPA) in the United States [2], and Canada's PIPEDA privacy legislation [29], grant individuals the right to know how their data is being used. The growing trend of legal disputes over unauthorized data use in ML and the legislation of data protection regulations highlight an urgent need for reliable data-use auditing to ensure accountability and transparency in ML models, addressing both legal and ethical concerns.

Data-use auditing of ML models: Data-use auditing is a technique that a data owner can use to verify whether her published data has been used in the training of an ML model. This approach can be broadly categorized into two levels: dataset-level [39, 43, 51, 62, 80, 81, 93, 118, 142, 163, 168] and instance-level [15, 71, 83, 107, 121, 126, 128, 171, 178] data-use auditing. Dataset-level data-use auditing is applied in scenarios where a data owner possesses a substantial dataset. It produces an auditing result for the entire dataset, by detecting a significant signal in an ML model that requires learning from multiple data samples [80, 118] or aggregating information across individual instances [25, 62]. Instance-level data-use auditing addresses this limitation by offering fine-grained auditing results, assessing the use of just a few data instances in ML models.

Research gaps in auditing data use of ML models: Existing data-use auditing methods exhibit several fundamental limitations. First, current dataset-level auditing approaches are restricted to scenarios in which the whole training set of the ML model is contributed from

one data owner and thus the data owner has control over the whole dataset [51, 80, 81, 118, 142], including, e.g., knowledge of the labels [80, 81, 118]. This limits their application in a real-world setting where the training dataset might be collected from multiple data owners or data sources. In addition, the existing works focus on a particular type of ML model, e.g., image classifiers [80, 81, 118, 168], and do not directly generalize to other domains. These dataset-level auditing methods are also unsuitable for cases where the data owner has a small dataset or even a single data instance.

Second, existing instance-level data-use auditing methods are passive, requiring no modification to the audited data instance before its publication. They apply techniques originally developed for membership inference attacks [55, 128]. These techniques usually require access to auxiliary data sampled from the same distribution as the training data of the audited ML model and use them to train reference models (i.e., ML models similar to the audited model) [15, 128, 171, 178]. More critically, passive data-use auditing cannot provide guarantees on false-detection rates, rendering its detection results less reliable and convincing. These make passive data-use auditing less suitable for high-stakes auditing scenarios [181]. Taken together, these limitations reveal the absence of a general and statistically principled framework that enables reliable data-use auditing across diverse models and ownership settings, while providing rigorous false-detection guarantees.

While recent work has introduced various data-use auditing methods, their robustness against adaptive adversaries remains largely unexplored. Understanding whether and how such auditing mechanisms can be evaded is essential for characterizing their fundamental limits. This motivates the study of adversarial data purification strategies that deliberately weaken auditing signals, providing insight into the boundary between detectable and undetectable data use.

1.3 Dissertation Statement

In this dissertation, we argue that unauthorized data use—whether through credential-database compromise in authentication systems or the incorporation of datasets into ML

model training without the data owners' consent—can be reliably detected by data owners. Detecting these unauthorized data uses can be achieved by proactively introducing carefully designed randomness into data and combining anytime-valid statistical tests with system-aware threat modeling, enabling tunable and provably bounded global false-detection guarantees. We further show that these detection mechanisms can be mitigated somewhat using additional data, either from other sites in the case of credential-database compromise or from other data sources in the case of ML model training.

1.4 Contributions

To support this dissertation statement, the dissertation develops two anytime-valid detection frameworks for identifying unauthorized data use in authentication and machine learning systems, respectively. In addition, it provides the first systematic analysis of honeyword-based detection methods under a realistic cross-site threat model and introduces a data purification framework that mitigates data-use auditing in adversarial settings. Together, these contributions establish a principled foundation for anytime-valid detection of unauthorized data use and clarify the capabilities of statistical auditing in security-critical systems.

Analyzing the impact of exposed passwords on honeyword efficacy: In Chapter 3, we conduct the first critical analysis of honeyword methods in a threat model where the attacker knows legitimate passwords at other sites for the users represented in a database it is targeting. This threat model poses significant challenges to honeyword efficacy for user-chosen (versus algorithmically generated) passwords. Through a systematic analysis of current honeyword-generation algorithms, we quantify this tension and, by doing so, show that there appears to be no known algorithm providing a good tradeoff between false positives and false negatives for user-chosen passwords.

We then turn our attention to accounts with algorithmically generated passwords, as might be generated by a password manager. The critical finding that we uncover in this case is that honeyword-generation algorithms that do not take into account the method by which

the legitimate password was generated will yield high false-negative probability. We quantify the ability of the adversary to distinguish user passwords from honeywords against existing honeyword-generation algorithms, most of which do not guarantee honeywords of the same pattern as the legitimate password. We then consider the possibility that the honeyword-generation algorithm itself leverages a password manager to generate honeywords whenever the user does. In this case, if the attacker knows potentially more passwords for the same user's accounts elsewhere, it can classify the user's typical configuration better than the defender can, which implies an increase in false negatives.

Taken together, our results provide a cautionary note for the state of honeyword research and pose new challenges to the field.

LeakSentinel: a honeyword framework for anytime-valid credential-database breach detection: Base on the insights from our honeyword analysis that existing honeyword methods are prone to false alarms, and achieving both good flatness and a small false-detection rate is elusive, in Chapter 4, we propose LeakSentinel, a honeyword framework for anytime-valid credential-database breach detection. LeakSentinel is anytime-valid in that it guarantees a provably bounded global false-detection rate per site deployment, without requiring assumptions about attack intensity—that is, assumptions about how aggressively a false-alarm attacker issues login attempts over time and across accounts (e.g., the number of login attempts, the number of targeted accounts, and the frequency of such attacks). It consists of a honeyword-generation algorithm and a breach-detection algorithm. The honeyword-generation algorithm applies any password generative model within either a weighted-sampling-without-replacement process or a Bernoulli-sampling process, thereby subsuming most existing honeyword-generation methods. Our breach-detection algorithm operates online and leverages a sequential hypothesis test to continuously analyze incorrect login attempts without a predetermined termination time. Our breach-detection algorithm permits the specification of a parameter $\alpha < 1$ so that, in the absence of credential-database breach, the probability that it ever raises an alarm over an infinite sequence of login attempts is provably bounded by α. LeakSentinel enables effective detection of credential-database

breaches when a sufficient number of accounts are exploited by a breach attacker, even if the breach attacker leverages cross-site knowledge and applies adaptive adversarial strategies.

A general framework for data-use auditing of ML models: In Chapter 5, we turn to the problem of detecting unauthorized data use in training ML models. We propose a general, proactive method that supports both dataset-level and instance-level data-use auditing of ML models. Our approach consists of two key components: a data-marking algorithm and a data-use detection algorithm. The data marking algorithm, which the data owner applies prior to data publication, generates n (n can be any integer larger than one) distinct marked versions of a data instance by adding n unique marks (e.g., pixel alterations for images). Each marked version is carefully designed to preserve the utility of the original data instance while ensuring that the marked versions are maximally distinct. For the example of images, we could measure distinction by the distances between their high-level features prepared by a pretrained feature extractor. This marking process is agnostic to the ML task in which the marked data might be used (including, e.g., labels). After generating the n marked data, the data owner publishes only one version, selected uniformly at random, while keeping the remaining versions secret.

Once an ML model is accessible—even in only a black-box way—any "useful" membership inference method can be applied to measure the "memorization" score of each marked version, including the published one and those kept secret. If an ML model has not used the published marked data instance in training, then the rank of its "memorization" score relative to the unpublished versions should follow a uniform distribution over $\{1, 2, \ldots, n\}$ since we select it uniformly at random in the data-marking step. If, instead, the ML model has used the published marked data item in training, then its rank (based on its score) is more likely to be high, as the ML model tends to memorize its training data [134]. We develop a novel sequential method to estimate the summed ranks of the published data. This approach is anytime-valid, allowing the data owner to adaptively stop querying the audited ML model at any time while maintaining a controlled false-detection rate.

AcidWash: a framework to purify auditable training data: In Chapter 6, we study adversarial

data purification strategies that mitigate data-use auditing. We propose AcidWash, a novel and general framework to purify marked training data. Given a marked training dataset, a subset of which is detected as a reference dataset by an existing detector, AcidWash adds carefully crafted perturbations to the inputs in the remaining, untrusted dataset. In doing so, AcidWash aims to achieve three goals: 1) evasion, which is that AcidWash mitigates data-use auditing of a model trained on the purified training data; 2) utility, which is that a model learned on the purified training data is accurate; and 3) generality, which is that AcidWash is applicable to a wide range of data-use auditing algorithms.

To achieve the evasion goal, our key observation is that a model trained on the reference dataset is unlikely to have the auditable property since an overwhelming majority of the reference data are clean (assuming the detector is reasonably accurate). Therefore, AcidWash perturbs data in the untrusted dataset so that they follow a similar distribution to the reference dataset. We quantify the distribution similarity between the two datasets using the well-known Wasserstein distance [50, 149], due to its intuitive interpretation (the minimum cost of moving a pile of earth in the shape of one distribution to achieve the shape of the other). To achieve the utility goal, AcidWash aims to bound the perturbations added to the untrusted inputs. Formally, we formulate finding the perturbations as a minimization optimization problem, where the objective function minimizes the Wasserstein distance between the two dataset distributions (quantifying the evasion goal) under the constraint that the perturbation magnitudes are bounded (quantifying the utility goal). AcidWash achieves the generality goal by not depending on any specific data-use auditing algorithm, when formulating and solving the optimization problem.

1.5 Organization of the Dissertation

The rest of this dissertation is organized as follows. Chapter 2 describes the related work. Chapter 3 and Chapter 4 focus detecting credential-database breaches using honeywords. Chapter 3 introduces our honeyword analysis under realistic threat models where the breach attackers leverage cross-site knowledge and the false-alarm attackers attempt

to trigger false breach alarms without breaching the credential database. Chapter 4 introduces our honeyword framework, LeakSentinel, for anytime-valid credential-database breach detection. Chapter 5 and Chapter 6 address detecting unauthorize data use in ML model training. Chapter 5 introduces our general framework for data-use auditing of ML models. Chapter 6 proposes our data purification framework, AcidWash, for mitigating data-use auditing. Chapter 7 concludes the dissertation, summarizing the contributions, and identifying open research questions and opportunities for future work.

2. Related Work

Detecting unauthorized data use in authentication systems and ML systems has attracted significant attention in both security and machine learning research. In particular, prior work has studied mechanisms for detecting credential-database breaches in authentication systems and techniques for auditing whether datasets have been used in training ML models.

In this chapter, we review the literature most relevant to this dissertation. We first discuss prior work on detecting credential-database breaches using honeywords and their security analysis. We then review existing methods on auditing data use in ML models and potential attacks to them. This discussion provides the background necessary to understand the limitations of existing methods and motivates the works developed in the subsequent chapters.

2.1 Detecting Credential-Database Breaches

In this section, we review prior work on detecting credential-database breaches using honeywords. We organize the discussion into four categories: honeyword system architectures, honeyword-generation methods, breach-detection mechanisms, and existing analyses of honeyword-based detection.

2.1.1 Honeyword System

Existing honeyword-based authentication systems include the honeychecker-based system [69], ErsatzPasswords [5], Lethe [37], and Amnesia [162]. The honeychecker-based system proposed by Juels and Rivest [69] can distinguish the user password from honeywords at login time, by storing the index of the user password in an unbreachable component, known as the honeychecker. ErsatzPasswords [5] is another system design capable of distinguishing the user password from honeywords at login time. It uses an unbreachable machine-dependent function to detect the use of honeywords. Lethe [37] uses unbreachable synchronized random number generators to replay all login events in order to detect the submission of a honeyword. Amnesia [162] does not rely on any secret datum for honeyword detection. Instead, it maintains a binary marker for each sweetword, all of which are

assumed to be exposed to a breach attacker, and detects honeyword use based on marker values of the submitted sweetwords. Here a sweetword is either the user's true password or one of its honeywords. However, Lethe [37] and Amnesia [162] cannot distinguish user passwords from honeywords at login time. Consequently, these two systems allow the possibility that a breach attacker compromise an account using a honeyword instead of the actual user password.

Our honeyword framework introduced in Chapter 4, LeakSentinel, can be integrated into any honeyword system design in which the user password can be distinguished from other passwords, including honeywords, at login time. We adopt the classic honeychecker-based architecture as our default. We discuss how LeakSentinel can be integrated to ErsatzPasswords [5], and explain why it cannot be extended to Lethe [37] and Amnesia [162].

2.1.2 Honeyword Generation

Prior work on honeywords has primarily focused on the design of honeyword-generation techniques [4, 38, 44, 52, 157, 175]. Most of them leverage a password generative model [108, 170], either learned from a password dataset or heuristically constructed, to generate honeywords. For example, in the first honeyword proposal by Juels and Rivest [69], they introduced a chaffing-by-tweaking method that randomly replaces characters in the user password to generate honeywords. The corresponding password generative model is therefore an input-dependent, heuristically constructed distribution that assigns equal probability to a set of passwords obtained by tweaking the input password. Erguler [44] and Guo et al. [52] instead reused the existing user passwords in the system as honeywords, yielding a generative model that is simply the empirical distribution over stored passwords. Wang et al. [157] proposed to leverage a range of password generative models including list models [156], Markov models [90], and PCFG models [167] with auxiliary information incorporated (e.g., personally identifiable information (PII)). Bernoulli Honeywords [160] independently selects each password as honeyword with a constant probability and thus their password generative model is a random generator that assigns equal probability to all the passwords. More recently,

several works [175] have explored the use of large language models (LLMs) as password generative models in honeyword generation.

Our honeyword framework LeakSentinel, introduced in Chapter 4, consists of a honeyword-generation algorithm that is compatible with any password generative model (e.g., chaffing-by-tweaking model, Markov model, PCFG, or recurrent neural network (RNN)). Therefore, it subsumes most existing honeyword-generation techniques. We instantiate representative models in the implementation of our honeyword-generation algorithm.

2.1.3 Breach Detection

Existing works [44, 61, 69, 156, 157, 160] largely rely on a simple breach-detection mechanism that trigger an alarm when a honeyword or a sufficient number of honeywords is submitted in login time. This design choice is reflected in their evaluation metrics, e.g., flatness [61, 69, 160] and success-number graph [156, 157]. However, such mechanisms incur a false-detection rate that grows with the number of login attempts from false-alarm attackers. As such login attempts accumulate, this leads to a high and unbounded false-detection rate. In practice, this often causes operators to disregard or disable the detection mechanism altogether. Although some studies have discussed alarm policies to mitigate the cost of false alarms [4, 61, 69], it is critical to design a breach-detection algorithm that can provide a negligible, provably bounded false-detection rate and this problem remains open.

We address this gap with LeakSentinel (that introduced in Chapter 4), which incorporates a novel breach-detection algorithm based on a anytime-valid statistical test over incorrect login attempts. Combined with our honeyword-generation algorithm, LeakSentinel enables effective detection of credential-database breaches while guaranteeing a negligible, provably bounded global false-detection rate.

2.1.4 Honeyword Analysis

Recent work has investigated the security of honeywords under breach attack scenarios where the attacker possesses auxiliary information about the users or the password distribution at the targeted site. Such information can be leveraged to distinguish the legitimate

password from honeywords. Wang et al. [156] performed the first security analysis on honeywords under such attacks, but they focused only on the legacy-UI methods proposed by Juels and Rivest, empirically showing that these methods fail to achieve low false-negative rates. More recently, Wang et al. [157] additionally considered registration order (the time when the user accounts were created) as the auxiliary information available to the breach attacker for distinction. They proposed leveraging the auxiliary information (e.g., PII or registration order) in a password model like the list model, probabilistic context-free grammars (PCFG) [167], a Markov model [90], or a combination thereof, to generate honeywords. Their proposed methods achieved low false-negative rates under the threat model considered in their work [157].

However, none of the existing studies consider the scenario in which attackers possess cross-site knowledge, such as passwords leaked from other services used by the same user. This setting introduces fundamental challenges for honeyword security. In Chapter 3, we conduct the first systematic analysis of honeyword-based detection under such a cross-site threat model. Our empirical results show that existing honeyword-generation techniques, including those proposed by Wang et al., exhibit high false-negative probabilities in this setting. We are the first to systematically analyze this trade-off, demonstrating that existing honeyword-generation methods cannot achieve both low false-positive and low false-negative rates when attackers possess cross-site password knowledge.

2.2 Auditing Data Use of ML Models

In this section, we review prior work on auditing data use in ML model training. We organize the discussion into four categories: proactive data-use auditing methods, passive data-use auditing methods (i.e., membership inference), data watermarking techniques, and existing countermeasures to data-use auditing.

2.2.1 Proactive Data-Use Auditing in ML

Proactive data-use auditing involves modifying the to-be-audited data before they are published [40, 51, 80, 81, 118, 142, 163, 166, 168, 176, 187] and typically consists of a data-

marking algorithm and a data-use detection algorithm. When its data-marking algorithm leverages randomness to modify data to establish a distribution for a test statistic under the null hypothesis that the data has not been used, its data-use detection algorithm can provide a statistical guarantee on the false-detection rate [118]. Existing methods address only dataset-level data-use auditing. The dataset-level proactive data-use auditing methods are applicable only in scenarios where a data owner has a substantial dataset with a large number of data instances. They produce auditing results by aggregating information across individual instances [25] or detecting prominent signals in an ML model that requires learning from multiple data samples [80, 118].

For example, a line of works on dataset-level proactive data-use auditing of image classifiers [80, 81, 142] is based on backdoor attacks [49, 120], to enable a data owner to modify a substantial portion of her dataset and then detect its use by eliciting predictable classification results from the model. But these methods do not provide any guarantee on the false-detection rate. Radioactive data [118] audits the use of a dataset in image classifiers with a statistical guarantee on false-detection rate. It works by embedding class-specific marks into a subset of the dataset and analyzing correlations between parameters of the final layer of the audited image classifier and the embedded marks. More recently, Chen and Pattabiraman [25] proposed a method that can be applied to audit the use of a small number of data samples in training image classifiers, but their method does not bound the false-detection rate.

In addition, the existing methods are domain-specific. They are tailored to particular ML tasks, e.g., image classifiers [25, 80, 118], language models [16, 166], or text-to-image generative models [78, 163]. Consequently, their data-marking algorithms require prior knowledge of the ML task for which the data might be used. In Chapter 5, we address this gap by proposing a general method that supports both dataset-level and instance-level data-use auditing across various domains. The data-marking algorithm of our approach does not require prior knowledge of the ML task for which the data might be used and our method provides a tunable, provably bounded false-detection rate.

2.2.2 Membership Inference in ML

Membership inference (MI) is a type of confidentiality attack in machine learning, which aims to infer if a particular data sample [15, 27, 59, 128, 171] or any data associated with a specific user [22, 97, 135] has been used to train a target ML model. The existing MI methods can be classified into shadow model-based attacks [85, 128] and metric-based attacks [119, 121, 137, 172]. Shadow model-based attacks leverage shadow models (i.e., models trained on datasets that are similar to the training dataset of the target model) to imitate the target model and so incur high costs to train them. In contrast, metric-based attacks leverage metrics that are simple to compute (e.g., entropy of the confidence vector output by the target classifier [121, 137]) while achieving comparable inference performance [119, 121, 172]. MI has been explored for various model types, e.g., image classifiers [119, 121, 137, 172], visual encoders trained by self-supervised learning [83], language models [107], reinforcement learning [39], and facial recognition models [22].

MI can be used as a passive instance-level data-use auditing method that a data owner can use to infer if her data is used in an ML model. However, such a passive method does not provide any quantitative guarantee for its inference results. Our framework introduced in Chapter 5 uses metric-based MI to measure "memorization" scores of marked data in the detection algorithm that provides a quantifiable, tunable guarantee on false detection.

2.2.3 Data Watermarking

Data watermarking is a technique used to track digital data by embedding a watermark that contains identifying information of the data owner. A classical example of image watermarking is zero-bit watermarking [19] that embeds information into the Fourier transform of the image. However, this type of traditional watermarking is not robust to data transformation. Recently, there have been research efforts on training deep neural networks (DNNs) to embed and recover watermarks that are robust to data transformation [8, 89, 141, 185]. DNN-based data watermarking is widely applied to attribute AI-generated content [45, 176].

Data watermarking can be used to audit data use to train a generative model [176], since

the watermark embedded in the training images could be transferred to the images generated from the model. However, this technique cannot be directly applied to other types of ML models, e.g., an image classifier. In contrast, instead of recovering the embedded marks from the ML model, our proposed auditing method in Chapter 5 detects the use of published data by analyzing the outputs of the ML model on the published data and the unpublished data.

2.2.4 Countermeasures to Data-Use Auditing

While recent research has focused on designing new data-use auditing algorithms, an open question is their robustness in adversarial settings, where countermeasures may be applied to evade the detection by a data-use auditing method. In this section, we review existing methods that have been considered by previous works as countermeasures to data-use auditing, as well as those that could potentially be adapted for this purpose. These methods can be broadly categorized into robust learning, detection, and purification.

Robust learning: This type of countermeasure to data-use auditing aims to redesign the ML training algorithm so that the learned model is guaranteed to be free of the hidden, auditable properties even if some training examples are marked. Examples of robust learning methods include randomized smoothing [76, 117, 183] and differential privacy [42, 58, 91]. Randomized smoothing adds random noise to the training inputs when learning a model, while differential privacy adds random noise to the gradients used to update the model during training. However, these robust learning methods require modifying the ML training algorithm and often sacrifice model accuracy [62].

Detection: A detection mechanism [111, 180] aims to detect the clean/unmarked training data in the marked dataset and only use those data for model training. While the existing detection methods were proposed as defenses against data poisoning attacks [48, 124], we adapt them to evade data-use auditing. For instance, Peri et al. [111] proposed a k-nearest-neighbors (k-NN) based method, which detects a training example as clean if its label is the same as the majority of its k nearest neighbors. Zeng et al. [180] proposed Meta-Sift, which formulates and solves a bilevel optimization problem to split the training dataset into

a clean set and a contaminated set such that a model trained on the clean set has the highest prediction loss when tested on the contaminated set.

In Chapter 6, we introduce a purification method, AcidWash, that complements existing detection methods. Rather than discarding the remaining training data that are not identified as clean by a detection method, our approach purifies them such that an ML model can be trained jointly on the detected data and the purified data. In Chapter 6.3, we leverage the k-NN detector to identify clean data that serves as the reference data in our purification method. This setup empirically demonstrates the effectiveness of our approach in real-world settings.

Purification: Purification aims to purify the training data detected as marked, so the training data detected as clean by a detector and the purified version of those detected as marked can be used together for model training. Unfortunately, purifying training data is largely unexplored. To the best of our knowledge, Friendly Noise [84] is the only training-data purification method, originally designed to defend against data poisoning attacks [48]. Specifically, Friendly Noise aims to add as large perturbations to the training inputs as possible without changing their predictions made by a model pre-trained on the original marked training data. However, the model learned on the original marked training data is different from the one learned on the purified training data. As a result, Friendly Noise achieves a suboptimal trade-off between model accuracy and effectiveness at eliminating the hidden properties from the learned model, as shown in our experimental results in Chapter 6.

Some methods [103, 173] purify testing inputs, e.g., adversarial examples. These methods require a large amount of clean training data to learn generative models (e.g., diffusion models [132]), which are subsequently used to purify testing inputs. They are not applicable in the problem of mitigating data-use auditing where the training data are marked.

3. The Impact of Exposed Passwords on Honeyword Efficacy

In this chapter, we conduct the first critical analysis of honeyword-generation algorithms in the setting where the breach attacker knows legitimate passwords at other sites for the users represented in a database he is targeting. There is reason to suspect that this threat model would pose significant challenges to honeyword efficacy for user-chosen (versus algorithmically generated) passwords. On the one hand, if the honeyword-generation algorithm used to populate the targeted database generates honeywords that are all dissimilar from the user-chosen password, then the known password(s) for the same user might enable the attacker to distinguish the user-chosen password from its honeywords with high probability. If so, the false-negative probability (the probability that the site fails to detect the breach) would be high. On the other hand, if the honeyword-generation algorithm generates some honeywords that are similar to the user-chosen password, then this might make it easier for an attacker who has *not* breached the database to guess and enter honeywords in login attempts, thereby inducing a false breach alarm (false positive).

Through a systematic analysis of current honeyword-generation algorithms, we quantify this tension and, by doing so, show that there appears to be no known algorithm providing a good tradeoff for accounts with user-chosen passwords. We additionally applied two password tweaking techniques from password guessing to improve honeyword generation. While these two algorithms relieve this tension by providing slightly lower false-negative probability, they still induce a high false-positive probability. Therefore, it remains far from clear that there is *any* honeyword-generation algorithm that ensures low false-negative probability and provides adequate resistance to false breach alarms (i.e., a false-positive rate near 0).

We then turn our attention to accounts with algorithmically generated passwords, as might be generated by a password manager. The critical finding that we uncover in this case is that honeyword-generation algorithms that do not take into account the method by which the legitimate password was generated will yield high false-negative probability.

For example, if the user employs a password manager that generates passwords to fit a user-configured specification, and if the passwords exposed for that user permit the attacker to infer this specification, then the attacker can discard any honeywords not fitting that pattern. We will quantify the ability of the adversary to do so against existing honeyword-generation algorithms, most of which do not guarantee honeywords of the same pattern as the legitimate password. We then consider the possibility that the honeyword-generation algorithm itself leverages a password manager to generate honeywords whenever the user does. However, due to the numerous generator configurations that users might adopt, doing so is not foolproof. In particular, if the attacker knows potentially more passwords for the same user's accounts elsewhere, it can classify the user's typical configuration better than the defender can. This advantage thus implies an increase in false negatives, which we will demonstrate in certain cases.

To summarize, our contributions in this chapter are as follows:

- We formalize the false-positive and false-negative rates of honeywords in a model in which the attacker possesses passwords for the same user at other sites (obtained by, e.g., breaching those sites or phishing the user).
- Using these definitions and empirical datasets of compromised passwords, we show that existing honeyword-generation algorithms (and two honeyword-generation methods adapted from password-guessing attacks) exhibit poor tradeoffs between false negatives and false positives in this threat model. All the analyzed methods have a false-negative rate much higher than random guessing (i.e., it is often easy for false-negative attackers to distinguish the account password from honeywords) or a false-positive rate much higher than zero (i.e., it is often easy for false-positive attackers to induce false breach alarms).
- We conduct the first study of using honeywords to protect algorithmically generated passwords. Though relevant only for sites that reversibly encrypt their password databases (since password hashing, which is best practice, should render algorithmically generated passwords irrecoverable to an attacker who breaches the database),

our study provides interesting findings in this setting. Using passwords gathered from popular password managers, we show that introducing honeywords without attention to the account's password being algorithmically generated offers little protection for existing honeyword-generation algorithms. We further explore the use of automatic password generators to generate honeywords when the account password is identified as being algorithmically generated itself, but find that the myriad configurations of these generators can be a pitfall for honeyword generation.

This chapter has been published on the 2024 USENIX Security Symposium [61].

3.1 Background

3.1.1 Definitions

Honeywords are decoy passwords added to each account entry in a credential database. The principle behind honeywords is that since the legitimate user does not know the honeywords generated for her account, the only party who is able to enter those honeywords is an attacker who discovered them by breaching the credential database. As such, login attempts using honeywords should be taken as compelling evidence of a database breach.

To make this principle precise, we define the false-positive and false-negative probabilities of a honeyword scheme in a way that abstracts away the details of the system leveraging them. We do so using the experiments shown in Fig. 3.1 and described in text below. In these experiments, a random user is modeled by a randomized algorithm gen, by which the user selects her password $x \in \mathcal{X}$ for a site. The invocation $\mathsf{gen}()$ outputs not only the password x, but also auxiliary information $U \subset \mathcal{X}$ that is correlated with x and that the attacker might learn. In this work, U will be passwords set by the same user at other sites, though other works have considered other types of auxiliary information (e.g., [156]). Given x, the site selects honeywords for this account using the randomized algorithm $\mathcal{H}_k$, which outputs a set H where $|H| = k$ and $x \notin H$.

A *false-positive attacker* $\mathcal{F}$ attempts to trigger a breach alarm at this site even though it has not breached the site, by leveraging its knowledge of x and $\mathcal{H}_k$ to guess honeywords in

Experiment $\mathsf{Expt}^{\mathrm{FPP}}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F})$
  $(x, U) \leftarrow \mathsf{gen}()$
  $H \leftarrow \mathcal{H}_k(x)$
  $F \leftarrow \mathcal{F}(x)$
  if $|F| \leq \gamma$
    $\wedge\ |F \cap H| \geq \tau$
    then return 1
    else return 0

$\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F}) \stackrel{\text{def}}{=}$
  $\mathbb{P}\Big(\mathsf{Expt}^{\mathrm{FPP}}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F}) = 1\Big)$
$\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma} \stackrel{\text{def}}{=}$
  $\max_{\mathcal{F}} \mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F})$

(a) False-positive probability

Experiment $\mathsf{Expt}^{\mathrm{FNP}}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B})$
  $(x, U) \leftarrow \mathsf{gen}()$
  $H \leftarrow \mathcal{H}_k(x)$
  $B \leftarrow \mathcal{B}(H \cup \{x\}, U)$
  if $|B \cap H| < \tau$
    $\wedge\ x \in B$
    then return 1
    else return 0

$\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B}) \stackrel{\text{def}}{=}$
  $\mathbb{P}\Big(\mathsf{Expt}^{\mathrm{FNP}}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B}) = 1\Big)$
$\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau} \stackrel{\text{def}}{=}$
  $\max_{\mathcal{B}} \mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B})$

(b) False-negative probability

FIGURE 3.1: Measures for breach detection by honeywords. A false-positive attacker $\mathcal{F}$ tries to trick an unbreached site into detecting that it has been breached (Fig. 3.1a). A false-negative attacker $\mathcal{B}$ attempts to access an account after breaching the site, without alerting the site to its breach (Fig. 3.1b).

H. In this work, we consider the worst case where $\mathcal{F}$ is permitted to know x since $\mathcal{F}$ might represent a legitimate user of this site or because it might represent an outsider who, say, phished x. $\mathcal{F}$ might know U but U does not help in guessing H if x is already known. $\mathcal{F}$ is provided knowledge of the honeyword-generation algorithm $\mathcal{H}_k$ to provide a conservative analysis.[1] $\mathcal{F}$'s probability of triggering an alarm is defined in Fig. 3.1a, where $\tau \geq 1$ is the number of honeywords whose entry will trigger a breach alarm and where $\gamma \geq 1$ denotes the number of login attempts $\mathcal{F}$ is permitted to attempt for this account. In words, given x (along with gen, $\mathcal{H}_k$, τ, and γ, which are public parameters of the experiment), $\mathcal{F}$ wins by outputting a set F that it can enter in its budget of login attempts ($|F| \leq \gamma$) and that will trigger an alarm ($|F \cap H| \geq \tau$). Traditionally the threshold for raising a breach alarm has

[1] Allowing $\mathcal{F}$ knowledge of $\mathcal{H}_k$ conforms with general security design principles; e.g., "Do not rely on secret designs, attacker ignorance, or *security by obscurity*." [148, p. 21]. In our context specifically, if the attacker knows only that $\mathcal{H}_k$ is one of several alternatives, it can try each alternative via a different account. In this sense, our measure is analogous to the *min auto* approach to measuring password strength [101, 147], in which the strength of a password is measured by the number of tries to guess it, under the guessing strategy (from among several) that minimizes that number.

typically been set to $\tau = 1$, though this definition permits other values; A larger τ implies a more stringent condition for raising an alarm. $\mathcal{F}$'s *false-positive probability* $\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F})$ is the probability that $\mathcal{F}$ wins, and the overall false-positive probability $\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}$ is $\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F})$ for the attacker algorithm $\mathcal{F}$ that maximizes that probability. When the parameters gen, $\mathcal{H}_k$, τ, and γ are clear from context, we will abbreviate $\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}(\mathcal{F})$ and $\mathsf{FPP}_{\mathsf{gen},\mathcal{H}_k,\tau,\gamma}$ as $\mathsf{FPP}(\mathcal{F})$ and FPP, respectively, to simplify notation.

In contrast, a *false-negative attacker* $\mathcal{B}$ is an attacker who attempts to access this user's account after breaching the site but without alerting the site that it has been breached. This adversary's advantage in doing so is defined in Fig. 3.1b. In words, $\mathcal{B}$ obtains the set $H \cup \{x\}$, sometimes called the *sweetwords* for this account, as well as auxiliary information U. The set $H \cup \{x\}$ is sweetwords are recovered by the attacker from the salted hash file. $\mathcal{B}$ then wins if it outputs a set B that will not trigger an alarm ($|B \cap H| < \tau$) and that permits it to access the account ($x \in B$). Here we presume that $B \subseteq H \cup \{x\}$, since passwords other than the sweetwords outside $H \cup \{x\}$ offer no help for $\mathcal{B}$ to achieve his goals. Consequently, we drop γ as a parameter of the experiment; since $\gamma \geq \tau \geq |B|$, it does not constrain $\mathcal{B}$'s choice of B. Again, traditionally the threshold for raising a breach alarm has been set to $\tau = 1$, in which case the probability with which $\mathcal{B}$ guesses x from the sweetwords $H \cup \{x\}$ on the first try (i.e., $|B| = 1$) is called the *flatness* of the honeyword scheme. $\mathcal{B}$'s *false-negative probability* $\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B})$ is the probability that $\mathcal{B}$ wins, and the overall false-negative probability $\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}$ is $\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B})$ for the attacker $\mathcal{B}$ that maximizes that probability. When the parameters gen, $\mathcal{H}_k$, and τ are clear from context, we will abbreviate $\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}(\mathcal{B})$ and $\mathsf{FNP}_{\mathsf{gen},\mathcal{H}_k,\tau}$ as $\mathsf{FNP}(\mathcal{B})$ and FNP, respectively, to simplify notation.

A honeyword-generation algorithm $\mathcal{H}_k$ can at best achieve $\mathsf{FPP} \approx 0$ and $\mathsf{FNP} = \frac{\tau}{k+1}$. Our research evaluates the extent to which known honeyword-generation algorithms, described in Sec. 3.1.2, approach this ideal. When considering false-negative attackers, we will evaluate an attacker who prioritizes accounts by its perceived likelihood of success in guessing the account password x, by refining gen to represent "easy" users for whom $(H \cup \{x\}) \cap U \neq \emptyset$, likely due to exact password reuse across accounts; "medium" users who are not "easy" but

for whom there are elements of $H \cup \{x\}$ and U that are close to one another (in a sense we will define later), likely because the user set passwords at her other accounts that are similar to x (partial password reuse); or "hard" users for whom neither condition holds.

3.1.2 Honeyword-Generation Algorithms

In this section, we introduce honeyword-generation algorithms, some of which have been introduced in previous works [38, 69, 157]. These honeyword-generation algorithms use a generative password model to generate honeywords [38, 157] by sampling candidates from the generative model as honeywords. A generative password model is a probabilistic model that assigns a probability to each password candidate from a password space $\mathcal{X}$ that includes all the possible passwords allowed by a site, e.g., passwords of length between 4 and 30. Such a probability distribution could be learned from a password dataset [90, 157, 167, 169] or heuristically constructed [69]. The distribution may be either independent of input or conditioned on information (e.g., user password or PII). Accordingly, honeyword-generation algorithms can be classified into two groups: *input-independent* algorithms and *input-dependent* algorithms.

3.1.2.1 Input-independent Honeyword Generation

Input-independent honeyword-generation algorithms generate honeywords independently of the user passwords. In other words, their used password generative models are independent of the user passwords. We consider four widely used password generative models: list model [156], probabilistic context-free grammar model (PCFG) [167], Markov model [90], and recurrent neural network (RNN) [96], and their combination [157]. We denote these generation methods as list, pcfg, mkv, rnn, and cbm, respectively.

List model (list*)*: The list model estimates the probability of password x' by $\mathbb{P}_{\mathsf{list}}(x') = \frac{\mathsf{num}(x')}{\mathsf{num}(\Sigma^*)}$, where $\mathsf{num}(x')$ denotes the occurrences of x' in the training set and $\mathsf{num}(\Sigma^*)$ denotes the total number of training samples.

Probabilistic context-free grammar model (pcfg*)*: This generative model treats a password as a sequence of segments, each a digit segment, letter segment, or symbol segment. Denoting

the alphabet $\Sigma = \mathsf{letters} \cup \mathsf{digits} \cup \mathsf{symbols}$ where

$$\mathsf{letters} = \{a, \ldots, z, A, \ldots, Z\} \qquad \overline{\mathsf{letters}} = \Sigma \setminus \mathsf{letters}$$

$$\mathsf{digits} = \{0, \ldots, 9\} \qquad \overline{\mathsf{digits}} = \Sigma \setminus \mathsf{digits}$$

$$\mathsf{symbols} = \{!, @, \#, \ldots\} \qquad \overline{\mathsf{symbols}} = \Sigma \setminus \mathsf{symbols}$$

PCFG models the probability $\mathbb{P}_{\mathsf{pcfg}}$ ("bike123") of a password "bike123", for example, as the product $\mathbb{P}(\mathsf{L}_4\mathsf{D}_3)\mathbb{P}\big(\text{bike} \mid \mathsf{L}_4\big)\mathbb{P}\big(123 \mid \mathsf{D}_3\big)$ where

$$\mathbb{P}(\mathsf{L}_4\mathsf{D}_3) = \frac{\mathsf{num}(\mathsf{letters}^4\mathsf{digits}^3)}{\mathsf{num}(\Sigma^*)}$$

$$\mathbb{P}\big(\text{bike} \mid \mathsf{L}_4\big) = \frac{\mathsf{num}(\text{bike} \cup \text{bike}\overline{\mathsf{letters}}\Sigma^* \cup \Sigma^*\overline{\mathsf{letters}}\text{bike}\overline{\mathsf{letters}}\Sigma^* \cup \Sigma^*\overline{\mathsf{letters}}\text{bike})}{\mathsf{num}(\mathsf{letters}^4 \cup \mathsf{letters}^4\overline{\mathsf{letters}}\Sigma^* \cup \Sigma^*\overline{\mathsf{letters}}\mathsf{letters}^4\overline{\mathsf{letters}}\Sigma^* \cup \Sigma^*\overline{\mathsf{letters}}\mathsf{letters}^4)}$$

$$\mathbb{P}\big(123 \mid \mathsf{D}_3\big) = \frac{\mathsf{num}(123 \cup 123\overline{\mathsf{digits}}\Sigma^* \cup \Sigma^*\overline{\mathsf{digits}}123\overline{\mathsf{digits}}\Sigma^* \cup \Sigma^*\overline{\mathsf{digits}}123)}{\mathsf{num}(\mathsf{digits}^3 \cup \mathsf{digits}^3\overline{\mathsf{digits}}\Sigma^* \cup \Sigma^*\overline{\mathsf{digits}}\mathsf{digits}^3\overline{\mathsf{digits}}\Sigma^* \cup \Sigma^*\overline{\mathsf{digits}}\mathsf{digits}^3)}$$

The pcfg model is trained by assigning the conditional probability of each production rule such that the likelihood of passwords in the training dataset is maximized.

*Markov model (*mkv*)*: This model is a Markov chain of a high-order Markov process (we set the order as 4) such that the prediction of character at the current position only depends on the characters at the previous 4 positions. That is, for a password $x = c_1 \ldots c_d$, $\mathbb{P}_{\mathsf{mkv}}(x') = \prod_{z=1}^{d} \mathbb{P}\big(c_z \mid c_{z-\min\{z-1,4\}\ldots c_{z-1}}\big)$ where

$$\mathbb{P}\big(c_z \mid c_{z-\min\{z-1,4\}} \ldots c_{z-1}\big) = \frac{\mathsf{num}(\Sigma^* c_{z-\min\{z-1,4\}} \ldots c_{z-1} c_z \Sigma^*)}{\mathsf{num}(\Sigma^* c_{z-\min\{z-1,4\}} \ldots c_{z-1}\Sigma^*)}$$

The mkv model is trained to maximize the likelihood of passwords in a training password dataset.

*Recurrent neural network (*rnn*)*: An RNN is a type of deep neural network, where nodes are connected to form cycles. This property enables a recurrent neural network to process ordered information, which makes it one of commonly used deep neural networks used in text generation [140]. Previous works (e.g., [96]) have demonstrated state-of-the-art performance by RNNs on password cracking and password strength measurement, essentially

by estimating the probability of a password much like a Markov model does, but without the constraint of the Markov assumption. Specifically, the RNN rnn models such a password distribution: for a shifted password $x' = c_0 c_1 \dots c_d$ with the start-of-sequence $c_0 = \mathsf{sos}$ added at the start and the end-of-sequence $c_d = \mathsf{eos}$ added at the end, the RNN defines the probability of a password as:

$$\mathbb{P}_{\mathsf{rnn}}(x') = \prod_{z=1}^{d} \mathbb{P}_{\mathsf{rnn}}(c_z \mid c_0 \dots c_{z-1}) = \prod_{z=1}^{d} \left[\mathsf{rnn}(c_0 \dots c_{z-1})\right]_{c_z}$$

where $\left[\mathsf{rnn}(c_0 \dots c_{z-1})\right]_{c_z}$ denotes the c_z-th component of vector $\mathsf{rnn}(c_0 \dots c_{z-1})$.

We referred to previous work [96] to design a RNN model, which consists of one embedding layer, one Long Short-Term Memory (LSTM) network [57], and a fully-connected layer. The embedding layer works like a lookup table converting each character in $\Sigma \cup \{\mathsf{sos}, \mathsf{eos}\}$ into a fixed length vector of 128 dimensions. The LSTM network is a stack of three LSTM layers, each of which is a set of recurrently connected memory blocks used to process information; it accepts 128 input features and has 128 hidden features. The fully connected layer is a linear layer with 128-dimension inputs and 96-dimension outputs. The RNN model is trained by minimizing the cross-entropy loss in the training dataset. In our experiments, we trained the RNN model using the Adam algorithm [70] with a learning rate of 10^{-3}. We set the batch size to be 8192 and the number of training epochs to be 30.

A combined model (cbm*)*: This generative model is a combination of several password generative models, namely list, pcfg, and mkv. For a password x', it estimate the probability by $\frac{1}{3}\mathbb{P}_{\mathsf{list}}(x') + \frac{1}{3}\mathbb{P}_{\mathsf{pcfg}}(x') + \frac{1}{3}\mathbb{P}_{\mathsf{mkv}}(x')$.

3.1.2.2 Input-dependent Honeyword Generation

Input-dependent algorithms generate honeywords that are dependent on the user passwords, i.e., they leverage a input-dependent password-generative model. Some input-dependent algorithms generate honeywords whose strength is equal or similar to the input password x. These methods still leverage password generative models such as list, pcfg, mkv, rnn, or their combination but select a sampled candidate as a honeyword if and only if its strength

is equal to that of the input password. However, if the input password is weak, it might be difficult to generate k honeywords with equal password strength, under the hypothesis that user-chosen passwords follow a Zipf distribution (e.g., [155]). So, in this work, we relax this requirement so that a sampled candidate will be used as a honeyword if its length equals the length of the input password. We denote this algorithm for generating honeywords from list, pcfg, mkv, rnn, or a combined method by list*, pcfg*, mkv*, rnn*, and cbm*. Other input-dependent methods generate honeywords by using an input-dependent generative model to tweak the input password. Here we consider four types of generative models: random-replacement model, targeted password model, credential-tweaking model, and Large Language Model (LLM).

Random-replacement model: These techniques generate honeywords by randomly changing some characters of the input password or similar passwords. We consider chaffing-by-tweaking or cbt-v [69], which generates honeywords by randomly replacing the last v characters of the input password with characters of the same type; cbt* [38], which generates honeywords by similarly replacing all the characters; and chaffing-by-a-hybrid-model (chm [38]).

cbt-v tweaks the input password by randomly changing the last v characters into other characters of the same types (i.e., symbols, letters, or digits). For example, "bike123z" can be tweaked by cbt-3 method into "bike164T". cbt* is an improved method by Dionysiou et al. [38], which randomly replaces all the symbols of the input password based on the following strategy: lower-case each upper-case letter with probability 0.3, capitalize each lower-case letter with probability 0.03, and change each digit to a different, uniformly chosen digit with probability 0.05.

Chaffing-with-a-hybrid-model (chm) [38] leverages a FastText model [10] to search for 9 nearest neighbors of the input password. Then, for each output from FastText model (including the input password), it applies cbt* to tweak the password to generate $(k+1)/10$ passwords. As such, totally k passwords are returned as honeywords. We followed the implementations used in previous works (e.g., [38]) in the training of FastText. We used the

`train_unsupervised` function of the FastText library to perform unsupervised training of the FastText "skipgram" model, with the minimal number of word occurrences set to be 1, the minimal length of character-grams to be 2, and the number of epochs to be 500.[2]

Targeted password generative model: These password generative models learn a distribution of *password templates* [157]. Here a password template is a pattern describing passwords set by the same user at different sites, wherein common substrings are indicated in the template using a special tag `SpecialTag`. For example, the template "`SpecialTag` z" might be generated from "bike123z" and "bike123" if these passwords were set by the same user at two different sites. Password generative models like PCFG are pretrained on a multiset of password templates, as targeted password generative models. Then, honeywords are generated by sampling templates from the targeted password generative models and replacing `SpecialTag` in the templates with the input password. We denote these generation methods from list, pcfg, mkv, rnn, or a combined method by listTar, pcfgTar, mkvTar, rnnTar, and cbmTar.

Credential-tweaking model: Credential-tweaking models leverage DNNs to learn a password distribution conditioned on an input password. We consider a password-to-password model (ptp) [106] and a password-to-editing-path model (ptep) [106], which are adapted from similar constructions originally developed to crack passwords [106].

The password-to-password model (ptp) is a deep neural network that models a conditional distribution. That is, given a password input x, for a shifted tweaked password $x' = c_0 c_1 \dots c_d$ with the start-of-sequence c_0 added at the beginning and the end-of-sequence c_d added at the end, the deep neural network models:

$$\mathbb{P}(x' \mid x) = \prod_{z=1}^{d} \mathbb{P}(c_z \mid c_0 \dots c_{z-1}, x) = \prod_{z=1}^{d} \left[\mathsf{ptp}(c_0 \dots c_{z-1}, x)\right]_{c_z},$$

where $\left[\mathsf{ptp}(c_0 \dots c_{z-1}, x)\right]_{c_z}$ denotes the c_z-th component of vector $\mathsf{ptp}(c_0 \dots c_{z-1}, x)$.

The password-to-editing-path model (ptep) is a DNN that models the probability of an edit path $\mathsf{trans}_{x \to x'} \subseteq \mathsf{transforms}^*$ conditioned on an input password. Here the edit path

[2] https://fasttext.cc/docs/en/python-module.html

$\mathsf{trans}_{x\to x'}$ is a sequence of transformations from the password x to x'. Here $\mathsf{transforms}$ is a global set of transformation units, each of which takes the form of $w = (\mathsf{op}, \mathsf{char}, z)$. op is an edit operation selected among insertion ins, substitute sub, and deletion del. $\mathsf{char} \in \Sigma \cup \{\perp\}$ is a character from Σ to insert or to substitute, or a empty symbol $\perp$ used in deletion. z is the position index of the input password to be edited. For example, (sub, "a", 1) denotes the transformation that substitutes the first character c_1 of the password with "a". Specifically, given a password input x, for a shifted edit path $\mathsf{trans}_{x\to x'} = w_0 w_1 \dots w_{d'}$ with the start-of-sequence $w_0 = \mathsf{sos}$ added at the beginning, the ptep models the conditional distribution as:

$$\mathbb{P}\big(\mathsf{trans}_{x\to x'} \mid x\big) = \prod_{z'=1}^{d'} \mathbb{P}\big(w_{z'} \mid w_0 \dots w_{z'-1}, x\big) = \prod_{z'=1}^{d'} \big[\mathsf{ptep}(w_0 \dots w_{z'-1}, x)\big]_{w_{z'}},$$

where $\big[\mathsf{ptep}(w_0 \dots w_{z'-1}, x)\big]_{w_{z'}}$ denotes the $w_{z'}$-th component of vector $\mathsf{ptep}(w_0 \dots w_{z'-1}, x)$.

After training, the ptep method generates honeywords as follows: at the z'-th step, given a password input x, the start-of-sequence w_0, and a sequence of previously generated transformation units $w_1 \dots w_{z'-1}$, ptep generates the next transformation unit $w_{z'}$ by randomly sampling a candidate from $\mathsf{transforms} \cup \{\mathsf{eos}\}$ based on the conditional distribution defined by $\mathsf{ptep}(w_0 \dots w_{z'-1}, x)$. The edit path generation process stops when it completes d' transformation units or it samples the end-of-sequence. Then the method produces a honeyword by applying the generated edit path on the password x.

LLM: These techniques generate honeywords by querying a large language model like GPT-3 [12] with prompts based on the input password. We consider a recently proposed method, chunk-level GPT-3 (cgpt) [175]. This technique includes two steps. First, taking the account password as input, a password-specific segmentation technique called PwdSegment [169] is used to return the chunks contained in the account password. For example, given an input password "bike2000", PwdSegment returns two chunks: "bike" and "2000". Second, a prompt containing information of the account password and its chunks is provided to the GPT-3 model [12], e.g.,

`Derive` k `passwords that are similar to ‘‘bike2000’’ and contain ‘‘bike’’`

```
and ‘‘2000’’.  The length of the passwords should be at most 8.  Do not
add digits at the end of the passwords.
```

Then, the GPT-3 model returns a list of passwords, used as honeywords. The temperature of GPT-3 is set to one in order to guarantee diversity of the honeywords generated, as suggested by Yu and Martin [175]. However, the number of honeywords output by GPT-3 is usually unequal to k, especially when k is large (e.g., $k = 99$). If the number of returned passwords is more than k, we use the top k as honeywords. If the number is less than k, we use cbt^* to tweak the returned passwords one-by-one until the number of honeywords reaches k.

3.2 User-Chosen Passwords

The first case we consider is when gen is an algorithm implemented by an average human user, and U is a multiset of passwords chosen by the same user at other sites. In this case, we show that the field has yet to identify *any* honeyword-generation algorithm that achieves small FNP and FPP simultaneously. Intuitively, this is true because when a user selects passwords without automated help (i.e., gen is an average user), then an attacker who guesses passwords F that are similar to passwords in U will be highly effective in either inducing false detections (a high $\mathsf{FPP}(\mathcal{F})$) or avoiding true detection (a high $\mathsf{FNP}(\mathcal{B})$). On the one hand, if $\mathcal{H}_k(x)$ outputs honeywords dissimilar to x, then since users often choose x similar to elements of U, it will be relatively easy for an attacker $\mathcal{B}$ to select x from $H \cup \{x\}$ as the one most similar to passwords in U. So, for $\mathsf{FNP}(\mathcal{B})$ to be small, $\mathcal{H}_k(x)$ must output at least some honeywords that are similar to x. On the other hand, the more it does so, the easier it is for an attacker $\mathcal{F}$ to induce false detections by guessing passwords F similar to passwords in U.

3.2.1 Attack Strategies

In this section, we introduce the false-positive attacker $\mathcal{F}$ and the false-negative attacker $\mathcal{B}$ that we use in the evaluation of $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$, respectively.

False-positive attacker $\mathcal{F}$: In the evaluation of $\mathsf{FPP}(\mathcal{F})$, recall that $\mathcal{F}$ is given access to x.

The attacker $\mathcal{F}$ leverages the honeyword-generation algorithm $\mathcal{H}$ on input x to generate a set of honeyword candidates. Then, if applicable, it sorts the candidates by the probabilities assigned by the honeyword-generation algorithm and uses the top γ candidates as the guessed honeywords F; otherwise, it picks γ candidates uniformly at random without replacement as F.

False-negative attacker $\mathcal{B}$: We evaluate $\mathsf{FNP}(\mathcal{B})$ for user-chosen passwords as follows. Given passwords U, $\mathcal{B}$ leverages a metric function to measure the similarity between the elements of U and the sweetwords $H \cup \{x\}$, and ranks each sweetword based on its similarity to the most similar element of U. The top τ ranked sweetwords are used to guess x.

3.2.2 Model to Measure Password Similarity

In the evaluation of $\mathsf{FNP}(\mathcal{B})$, we need to define a metric function that inputs a pair of passwords and returns a score reflecting the similarity between the inputs. To formulate such a metric function, we designed a similarity model $\mathsf{embed}(\cdot)$ by a deep neural network, which takes as input a password x and outputs its *latent representation* such that the cosine similarity between any two latent representations $\mathsf{embed}(x)$ and $\mathsf{embed}(x')$ grows with the probability that $\mathsf{gen}()$ would have output both, i.e., with $\mathbb{P}\big(x' \in U \mid (x, U) \leftarrow \mathsf{gen}()\big)$.

The similarity model is used to learn the embedding of passwords. Learning such an embedding of passwords into a latent space is essentially a *metric learning* problem [136, 159]. Therefore, we applied contrastive learning, which is one of the most widely used frameworks to train a model to perform this embedding so as to maximizing cosine similarity between positive (similar) pairs while minimizing cosine similarity of negative (dissimilar) pairs [23]. Training a contrastive model is performed in *batches*, each a multiset $\mathsf{batch} \subseteq \mathcal{X} \times \mathcal{X}$. Each $(x, x') \in \mathsf{batch}$ consists of similar passwords (intuitively, for which $\mathbb{P}\big(x' \in U \mid (x, U) \leftarrow \mathsf{gen}()\big)$ is high), whereas for any $(x'', x''') \in \mathsf{batch} \setminus \{(x, x')\}$, x and x'' are presumed to be dissimilar, as are x' and x'''. Training for a contrastive learning model of password similarity, therefore, updates embed to minimize a *loss function*, which typically

would take the form

$$\operatorname*{avg}_{(x,x')\in\mathsf{batch}} -\log \frac{\exp(\mathsf{sim}(\mathsf{embed}(x),\mathsf{embed}(x')))}{\sum_{\substack{(x'',x''')\in\mathsf{batch}:\\(x'',x''')\neq(x,x')}} \begin{pmatrix}\exp(\mathsf{sim}(\mathsf{embed}(x),\mathsf{embed}(x''))) + \\ \exp(\mathsf{sim}(\mathsf{embed}(x'),\mathsf{embed}(x'')))\end{pmatrix}} \tag{3.1}$$

where sim denotes cosine similarity (see Chen et al. [23]). Such updates with all the data samples from the training dataset passed through the trained model constitute one *epoch*.

We designed the similarity model by a transformer. The transformer-based similarity model embed consists of an embedding layer, a transformer encoder, and a fully connected layer. The embedding layer works like a lookup table converting each character into a 64-dimension vector. The transformer encoder network is a stack of three transformer encoder layers, each of which is composed by a four-head self-attention mechanism calculating the weighted average of 64-dimension representations and a fully-connected feed-forward network outputting 256-dimension outputs. The fully connected layer is a linear layer with 1920-dimension inputs and 256-dimension outputs.

We trained the similarity model to minimize Eq. (3.1) using the Adam algorithm with learning rate 10^{-3}. When training the model, each batch was created as a consecutive window of samples from the training sets. We found that with a batch size of $|\mathsf{batch}| = 256$, the loss Eq. (3.1) converged and the performance of the model became stable after only a few training epochs. Therefore, in all the results we report below, we trained the model for only 10 epochs.

3.2.3 Evaluation

In this subsection, we detail our evaluation of the user-chosen password case, which includes the used dataset, and the experimental results for $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$.

3.2.3.1 The Dataset

The dataset we used in the case of user-chosen passwords is the *4iQ* dataset [18], consisting of 1.4 billion (email, password) pairs, of which 1.1 billion emails and 463 million passwords are unique. Others attribute the *4iQ* dataset to various leaks from LinkedIn,

Myspace, Badoo, Yahoo, Twitter, Zoosk, and Neopet, and have used it to analyze users' choices of passwords across sites [106] (despite the possibility of some being automatically generated). Our use of leaked passwords was approved by our IRB, which specified protections in our handling of this data (who could access the data, what results could be reported, etc.). In order to use *4iQ*, we preprocessed the dataset by referring to previous works (e.g., [106]).

1) Cleaning: We removed any (email, password) pairs that satisfied any of the following conditions: the password contained non-ASCII characters, the space character, or a substring of 20 (or more) hex characters; the password had a length of less than 4 or more than 30; or the email contained non-ASCII characters or the space character.
2) Joining by email and username: For each email address appearing in the dataset, we collected the passwords appearing with that email address into a multiset. Then we merged some password multisets as follows: two multisets were merged if they contained at least one password in common and if the username parts of their email addresses were the same. We then eliminated each multiset containing only one password or $> 1{,}000$ passwords. In the resulting dataset, around 48% of users reused passwords, which is within the range between 43% and 51% estimated by previous work (e.g. [32]). More statistics about the resulting dataset are shown in Table 3.1.
3) Splitting into training and testing sets: Of the 195,894,983 password multisets that remained, 80% (156,722,455 multisets with 451,020,019 passwords) were set aside as training sets used to train models. The other 20% (39,172,528 multisets with 112,723,111 passwords) of the password multisets were set aside as testing sets. When evaluating $\mathsf{FNP}(\mathcal{B})$ and $\mathsf{FPP}(\mathcal{F})$, the algorithm gen was implemented by choosing x and the members of U without replacement from a single multiset chosen uniformly at random from the testing sets. Specifically, we randomly selected one password as the user password x and treated the remaining passwords as U. $|U|$ represents the amount of attacker's knowledge about this user's passwords at other sites. Its distribution in the test sets is shown in Fig. 3.2.

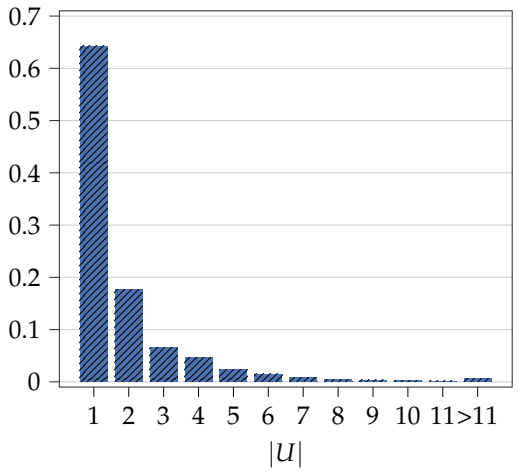


FIGURE 3.2: Distribution of $|U|$

Table 3.1: Statistics of the preprocessed password dataset

Statistic	Value
Total number of users	195,894,983
Total number of passwords	563,743,130
Average passwords per user	2.877
Average distinct passwords per user	1.961
Percentage of users reusing passwords	48.507%

3.2.3.2 Experimental Results

We now report $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ for the attackers $\mathcal{F}$ and $\mathcal{B}$ described in Sec. 3.2.1. To depict the tradeoffs between these measurements, we plot them against one another as τ is varied. When evaluating $\mathsf{FNP}(\mathcal{B})$, we isolate three subcases, to permit modeling of an attacker who prioritizes accounts based on similarities between $H \cup \{x\}$ and U per account. We measured such similarity based on definitions like those for password reuse introduced in previous work (e.g., [110]). Specifically, "easy" accounts are those for which $(H \cup \{x\}) \cap U \neq \emptyset$; "medium" accounts are those for which $(H \cup \{x\}) \cap U = \emptyset$ but there is a sweetword in $H \cup \{x\}$ that shares a substring of length at least four characters with some password in U; and "hard" accounts are those that are neither "easy" nor "medium". The percentages of accounts of different hardness are shown in Table 3.2.

Fig. 3.3 shows the tradeoffs between $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ for $k = 19$ honeywords and $\gamma = 1000$, for the various honeyword-generation algorithms described in Sec. 3.1. rnn and

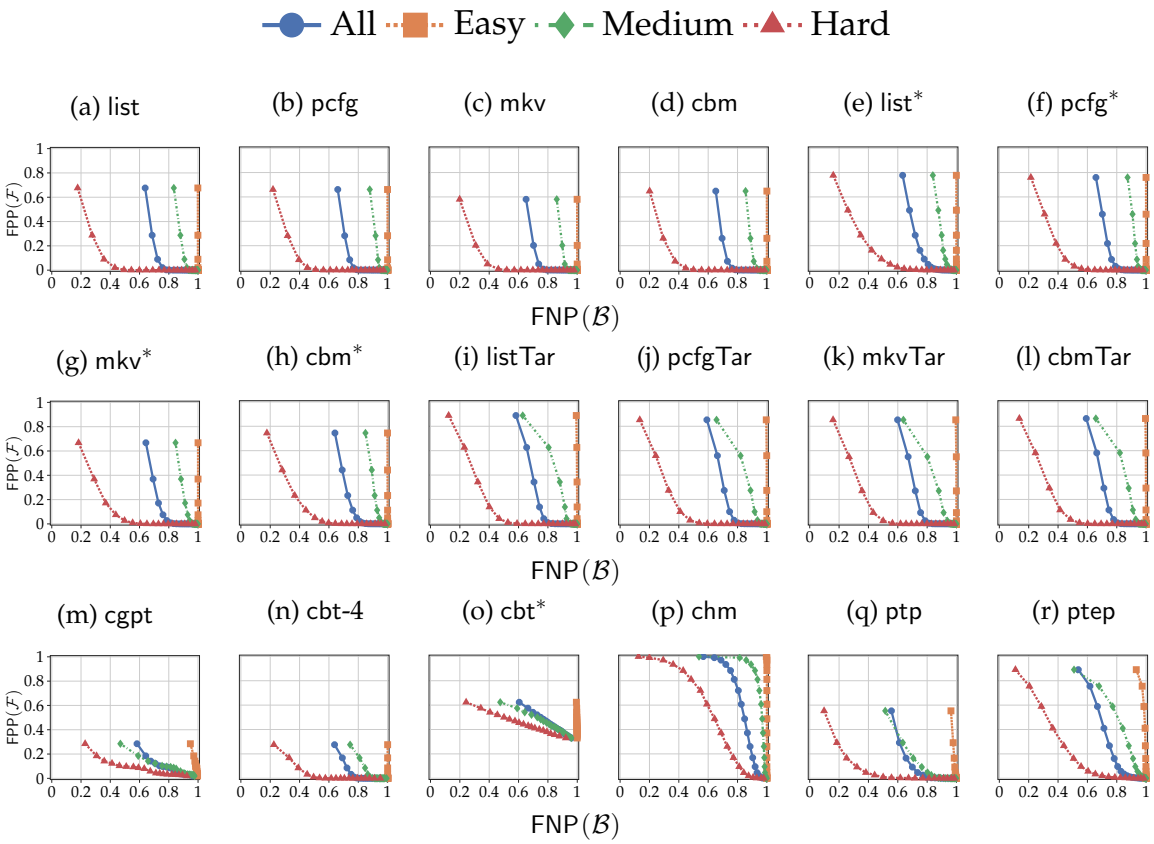


FIGURE 3.3: $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ as τ is varied, for the case of user-chosen passwords ($k = 19, \gamma = 1000$). The best $\mathsf{FNP}(\mathcal{B})$ are 0.54 (ptep, Fig. 3.3r), 0.56 (ptp, Fig. 3.3q), 0.57 (chm, Fig. 3.3p), and 0.58 (cgpt, Fig. 3.3m); all others have $\mathsf{FNP}(\mathcal{B}) > 0.59$. All suffer $\mathsf{FPP}(\mathcal{F}) > 0.27$ at $\tau = 1$. Those that reach $\mathsf{FPP}(\mathcal{F}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.81$.

its variants achieved similar performance to list, pcfg, mkv, cbm, and their variants, and thus we only show the results from list, pcfg, mkv, cbm, and their variants in Fig. 3.3. In each plot, there are four curves presenting the overall tradeoff ("all") and those of three subcases: "easy", "medium", and "hard". In each curve, markers highlight the $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ tradeoff at a specific values of τ ranging from $\tau = 1$ to k. Intuitively, a smaller τ yields lower $\mathsf{FNP}(\mathcal{B})$ but higher $\mathsf{FPP}(\mathcal{F})$ and so a marker closer to the top left corner. Increasing τ to k yields a higher $\mathsf{FNP}(\mathcal{B})$ but lower $\mathsf{FPP}(\mathcal{F})$ and so a marker closer to the bottom right corner. We stress that $\gamma = 1000$ yields an optimistic evaluation of $\mathsf{FPP}(\mathcal{F})$. For example, Florêncio, et al. [46] recommend that an account should withstand targeted online password-guessing attacks of 10^6 attempts in practice. As such, arguably $\gamma = 1000$

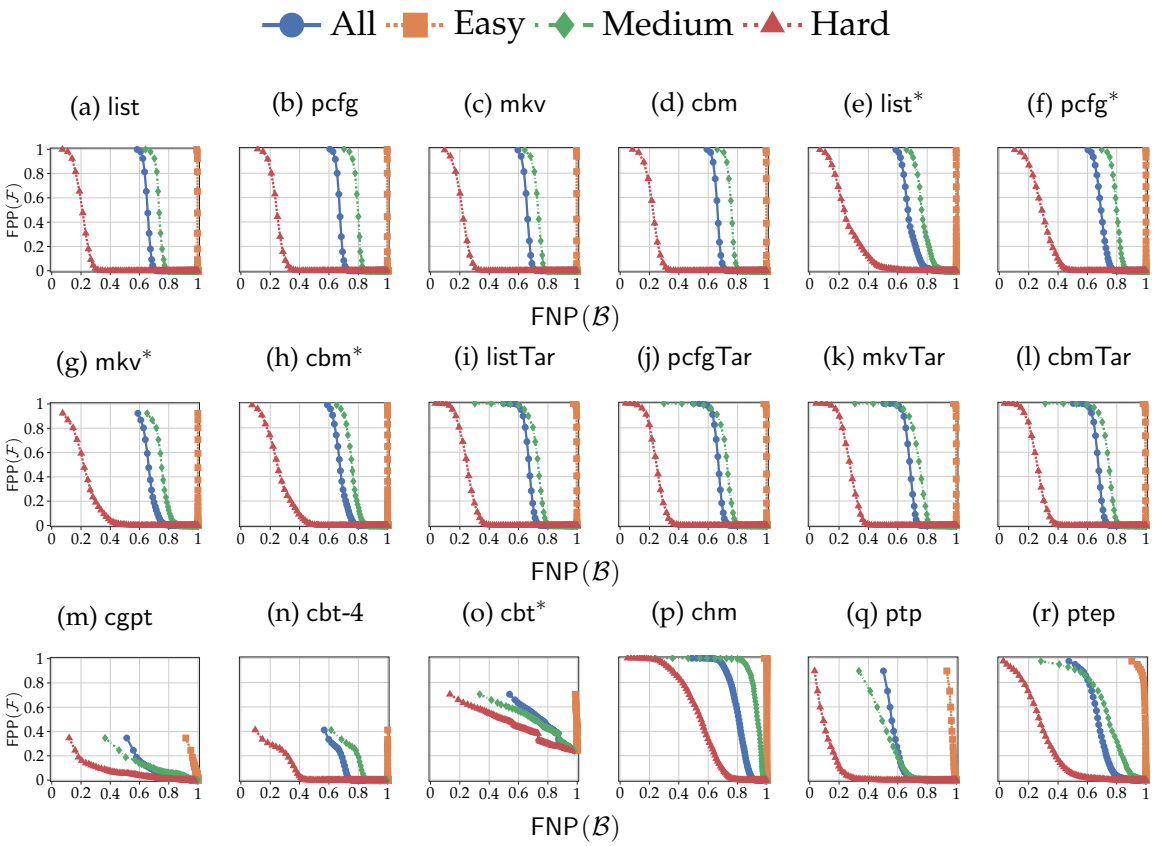


FIGURE 3.4: FPP($\mathcal{F}$) vs. FNP($\mathcal{B}$) as τ is varied, for the case of user-chosen passwords ($k = 99$, $\gamma = 1000$). The best FNP($\mathcal{B}$) are 0.47 (ptep, Fig. 3.4r), 0.49 (chm, Fig. 3.4p), and 0.50 (ptp, Fig. 3.4q); all others have FNP($\mathcal{B}$) > 0.50. All suffer FPP($\mathcal{F}$) > 0.34 at $\tau = 1$. Those that reach FPP($\mathcal{F}$) ≈ 0 do so with FNP($\mathcal{B}$) > 0.72.

is 1000× too small.

An ideal honeyword-generation algorithm would achieve FPP($\mathcal{F}$) ≈ 0 and FNP($\mathcal{B}$) $= \frac{1}{k+1}$ (which is 0.05 when $k = 19$) at $\tau = 1$. Unfortunately, no known honeyword algorithm comes close. As seen in Fig. 3.3, the best FNP($\mathcal{B}$) that the honeyword-generation techniques accomplish overall is 0.54 (ptep, Fig. 3.3r), 0.56 (ptp, Fig. 3.3q), 0.57 (chm, Fig. 3.3p), and 0.58 (cgpt, Fig. 3.3m); all others have FNP($\mathcal{B}$) > 0.59. When we consider the attacker prioritizing "easy" accounts, FNP($\mathcal{B}$) of ptep, cgpt, and ptp are at least 0.93, 0.95, and 0.96, respectively, while others have FNP($\mathcal{B}$) ≈ 1. This indicates that the false-negative attacker can break at least 43% accounts by targeting the "easy" ones, with only ptep, cgpt, and ptp presenting any significant chance of catching the attacker. That said, when such an

Table 3.2: Percentages of accounts of different hardness for a false-negative attacker $\mathcal{B}$, discussed in Sec. 3.2.3.2

	$k = 19$			$k = 99$		
$\mathcal{H}_k$	easy	med	hard	easy	med	hard
list	43.37	16.00	40.63	43.56	19.62	36.82
mkv	43.36	15.96	40.68	43.57	19.78	36.65
pcfg	43.36	15.59	41.05	43.47	18.40	38.13
rnn	43.36	16.01	40.63	43.59	19.57	36.84
cbm	43.33	15.98	40.69	43.55	19.27	37.18
list*	43.37	16.07	40.56	43.41	19.39	37.20
mkv*	43.33	16.05	40.62	43.38	19.57	37.05
pcfg*	43.35	15.66	40.99	43.38	18.68	37.94
rnn*	43.34	15.92	40.74	43.40	19.46	37.14
cbm*	43.37	15.94	40.69	43.38	19.70	36.92
listTar	44.11	19.25	36.64	43.55	15.80	40.65
mkvTar	44.00	18.91	37.09	43.47	15.92	40.61
pcfgTar	43.47	15.80	40.73	44.00	19.13	36.87
rnnTar	43.54	15.64	40.82	43.98	19.11	36.91
cbmTar	43.49	15.74	40.77	44.09	18.65	37.26
cgpt	44.54	14.16	41.28	44.98	14.28	40.72
cbt-3	43.35	14.87	41.78	43.34	15.28	41.38
cbt-4	43.33	14.97	41.73	43.33	15.31	41.36
cbt*	43.53	14.92	41.55	43.84	15.56	40.60
chm	43.46	15.33	41.21	43.91	15.83	40.26
ptp	44.80	14.22	40.98	45.89	14.67	39.44
ptep	46.05	12.36	41.59	47.39	12.18	40.43

attacker wants to guess more account passwords, i.e., targeting the "medium" accounts after the "easy" ones, the probability of inducing an alarm will increase with number of attacked accounts since $\mathsf{FNP}(\mathcal{B}) < 0.88$ for the "medium" subcase when $\tau = 1$. The four most successful algorithms (ptep, ptp, chm, and cgpt) are password-context-dependent techniques that generate honeywords similar to the account password, and thus it is more challenging for $\mathcal{B}$ to distinguish the account password from honeywords produced by these algorithms than from those of the other methods. We conclude that honeywords more similar to the account password yield a lower $\mathsf{FNP}(\mathcal{B})$, though one that is still far from $\frac{1}{k+1}$ due to password reuse.

However, ptep has $\mathsf{FPP}(\mathcal{F}) \approx 0.89$ at $\tau = 1$, where most others have lower $\mathsf{FPP}(\mathcal{F})$. The only exception is chm, which includes a deterministic step that searches for nearest neighbors of the account password and thus yields a high false-positive rate, $\mathsf{FPP}(\mathcal{F}) \approx 1$. While ptep is the best technique for generating honeywords similar to the account password, it is almost the easiest for the false-positive attacker to guess the generated honeywords with x known. Still, no generation method achieves $\mathsf{FPP}(\mathcal{F}) \leq 0.27$ at $\tau = 1$. Growing τ of course reduces $\mathsf{FPP}(\mathcal{F})$ but increases $\mathsf{FNP}(\mathcal{B})$: all methods capable of reaching $\mathsf{FPP}(\mathcal{F}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.81$ overall, $\mathsf{FNP}(\mathcal{B}) \approx 1$ for the "easy" subcase, and $\mathsf{FNP}(\mathcal{B}) > 0.91$ for the "medium" subcase.

A natural method to decrease $\mathsf{FNP}(\mathcal{B})$ would be to increase the number k of honeywords, but the more pronounced effect of doing so is increasing $\mathsf{FPP}(\mathcal{F})$, instead. Indeed, Fig. 3.4 shows the impact of increasing k to $k = 99$. As seen there, an order-of-magnitude increase in k resulted in a slight improvement to $\mathsf{FNP}(\mathcal{B})$ in each case, but a more substantial increase to $\mathsf{FPP}(\mathcal{F})$.

To summarize, honeyword-generation techniques like cbmTar that have been demonstrated to have good flatness in previous works (e.g., [157]) fail to achieve a low false-negative rate in our threat model, particularly not at settings of τ to ensure a small false-positive rate. Among the honeyword-generation techniques we consider, ptep achieves the best FNP but has a high FPP. Most other methods have lower FPP but a higher FNP. Regardless, in the case of user-chosen passwords, no existing algorithm achieves low rates of both false positives and false negatives. In addition, when the attacker targets the "easy" accounts that are approximately 43% of users, all the honeyword-generation methods are ineffective in detecting a breach at settings of τ achieving $\mathsf{FPP}(\mathcal{F}) \approx 0$.

3.3 Algorithmically Generated Passwords

The second case we consider is when gen is implemented using a password-generating algorithm. To our knowledge, this case has not been considered in prior honeyword research, and with good reason: Under the best practice of storing only preimage-resistant hashes of

passwords, it should be exceptionally difficult for an attacker who breaches a site's database to recover algorithmically generated passwords, due to their comparatively high strength. For this reason, algorithmically generated passwords in a breached credential database are primarily at risk if the hash function is less preimage-resistant than initially thought or when the site's database was reversibly encrypted—which, while not best practice, is necessary in some use cases (e.g., [98])—and the false-negative attacker recovered the decryption key along with the database.

We assume there is a large but limited number $\hat{n}$ of password generators denoted as $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$, each of which is defined by an algorithm and values of user-configurable parameters. We assume that each user determines gen by choosing a generator uniformly at random from $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$, and that each user stays with its choice. To justify this assumption, in Sec. 3.4.4 we report a brief study we did using the password policies of 20 commonly visited websites and Tranco Top 1M websites [113], where we found that setting passwords at these websites in a random order would permit the user to retain her chosen password-generation configuration for > 6.3 sites in expectation, before encountering a site for which the user's configuration was inconsistent. This finding is consistent with Alroomi et al. [6], who reported that only 15% sites have character constraints on password creation.

We assume the length of the generated passwords is one parameter that users can configure. Some password managers permit user configuration of allowable symbols, as well. Similarly, password managers that enable generation of easy-to-read passwords might avoid use of certain characters that are ambiguous in some fonts (e.g., "1" vs. "l" in sans-serif fonts). Password managers that generate easy-to-say passwords might restrict the symbols used in different positions of a password. We will see examples below. The user's choice of these parameters will generally be unknown to the defender, except as revealed by the account password x.

In this section, we analyze the contribution of honeywords for detecting credential database breaches for accounts with algorithmically generated passwords. We first show that honeyword-generation methods used in the user-chosen password case fail to achieve

both low false-negative rate and low false-positive rate for algorithmically generated passwords. Although utilizing password-generation algorithms to generate honeywords can do better, we show that the choice of selected generator is critical to achieving a low false-negative rate.

3.3.1 Attack Strategies

In this section, we introduce the false-positive attacker $\mathcal{F}$ and the false-negative attacker $\mathcal{B}$ used in the evaluation of $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$, respectively, when account passwords are generated algorithmically.

False-positive attacker $\mathcal{F}$: $\mathcal{F}$ uses the same strategy used in the case of user-chosen passwords in Sec. 3.2.1. Specifically, the attacker $\mathcal{F}$ leverages the honeyword-generation algorithm $\mathcal{H}$ to generate a set of candidates and sorts the candidates by their assigned probabilities, if applicable. Finally, it picks the top γ candidates as the guessed honeywords F.

False-negative attacker $\mathcal{B}$: $\mathcal{B}$ was implemented as follows. Given U, $\mathcal{B}$ leverages a classifier $\mathsf{classify}(\cdot)$ that outputs a confidence score per possible class. $\mathcal{B}$ classifies each element of U using $\mathsf{classify}$, using the highest-scored generator for each $x' \in U$ as a "vote" for the password generator that the user employs; the password generator obtaining the most such votes is denoted $\mathsf{gen}_{\hat{i}_{\mathcal{B}}}$. Then $\mathcal{B}$ assigns scores to the sweetwords from $H \cup \{x\}$ as follows: if the length of the sweetword is the same as those in U, $\mathcal{B}$ utilizes the classifier $\mathsf{classify}(\cdot)$ to value the sweetword by the confidence score of being from class $\hat{i}_{\mathcal{B}}$; otherwise, $\mathcal{B}$ will value it by 0. The attacker ranks the sweetwords based on the assigned scores and uses the top τ sweetwords as B.

We design the classifier used by $\mathcal{B}$ to consist of one embedding layer, a LSTM network, and two fully connected layers. The embedding layer converted each character into a vector of 128 dimensions. The LSTM network was a stack of three LSTM layers, accepting 128 input features and 128 hidden features. The first fully connected layer was a linear layer with 128-dimension inputs and 64-dimension outputs. The second had 64-dimension inputs and 96-dimension outputs. We trained the model by minimizing the cross-entropy loss using

the Adam algorithm [70] with a learning rate of 10^{-4}. We set the batch size to 1024 and the training epochs to 300. To constitute each training batch, we select a consecutive window of 1024 passwords from the algorithmically generated password training dataset (introduced in Sec. 3.3.3.1). We used classification accuracy as the evaluation metric. After each epoch, we evaluated the trained classifier on the algorithmically generated password evaluation dataset (introduced in Sec. 3.3.3.1) and saved the model that had the best performance on evaluation. The best model achieved evaluation accuracy of 90.91%. Increasing the amount of training data did not materially improve the accuracy.

3.3.2 Generating Honeywords Using Algorithmic Password Generators

The honeyword-generation methods introduced in Sec. 3.1.2 do not fare well (in terms of false-negative probability) when the account password is generated algorithmically. Intuitively, the input-independent honeyword generators fail to achieve a low $\mathsf{FNP}(\mathcal{B})$ since the honeywords they generate are user-chosen passwords, which makes it easy for $\mathcal{B}$ to distinguish the algorithmically generated account password from the honeywords. Many input-dependent generators do little better, because even though the account password is algorithmically generated, these models are trained on artifacts of human behavior, which renders the honeywords recognizable to $\mathcal{B}$. The primary exceptions are cbt-3 and cbt-4, which are not trained at all. These can achieve a low $\mathsf{FNP}(\mathcal{B})$, though still with a too-high $\mathsf{FPP}(\mathcal{F})$. We have empirically demonstrated these findings in Sec. 3.3.3.2.

Therefore, here we consider the use of algorithmic password generators to generate honeywords for algorithmically generated passwords submitted by the user. Given an account password x, the honeyword system selects a generator from $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$ and then leverages the selected generator to generate k honeywords. We categorize the methods based on the selection strategy, as follows:

- fxd: Given a fixed $\mathsf{gen}_{\hat{i}_*} \in \{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$, $\mathcal{H}_k$ samples k distinct honeywords using $\mathsf{gen}_{\hat{i}_*}$ to build H.
- rgm: $\mathcal{H}_k$ samples a $\mathsf{gen}_{\hat{i}}$ uniformly from $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$ and builds H by sampling k distinct

honeywords using $\mathsf{gen}_{\hat{i}}$.

- cls: $\mathcal{H}_k$ classifies the account password into one of $\hat{n}$ classes, indicating the generator $\mathsf{gen}_{\hat{i}}$ most likely to have generated it. $\mathcal{H}_k$ then builds H by sampling k distinct honeywords using $\mathsf{gen}_{\hat{i}}$.

3.3.3 Evaluation

3.3.3.1 Dataset

The datasets we used to evaluate honeyword-generation strategies in the case of algorithmically generated passwords were synthetically produced by querying online password generators.[3] Specifically, after browser-integrated password managers (Google Password Manager and iCloud Keychain), LastPass and 1Password are two of the most widely used password managers/generators [151]. LastPass permits the user to select one of three password-generation algorithms, namely "Easy-to-say", "Easy-to-read", or "All-characters". For each type, users can further specify the generator by checking or unchecking "Uppercase", "Lowercase", "Numbers", or "Symbols", though the "Easy-to-say" generator does not permit inclusion of Symbols or Numbers. 1Password allows users to select a type of password from among "Random Password", "Memorable Password", and "Pin". The Random Password generator includes Lowercase and Uppercase letters always, but users can check or uncheck Numbers and Symbols. The Memorable Password algorithm generates memorable passwords, each of which is a sequence of word fragments connected by separators. In this option, users can select separators among "Hyphens", "Spaces", "Periods", "Commas", "Underscores", "Numbers", and "Numbers and Symbols". In addition, users could check or uncheck "Full Words" and "Capitalize" to specify the "Memorable Password" generator. In this work, we used all the configurations from LastPass and 1Password's Random Password, and selected configurations for 1Password's Memorable Password. We consider passwords generated from each specification as one class, yielding 38 classes in total. These classes are shown in Table 3.3. We set the fixed $\mathsf{gen}_{\hat{i}_*}$ to be the "All characters" generator from

[3] We used PyAutoGui (`https://pyautogui.readthedocs.io/en/latest/`) to automate interactions with the password managers like 1Password and LastPass. That is, we automated generating random passwords, copying them into the clipboard, and storing them in a local file interactively.

(a) Honeyword generator $\mathcal{H}_k$ (b) Attacker $\mathcal{B}$

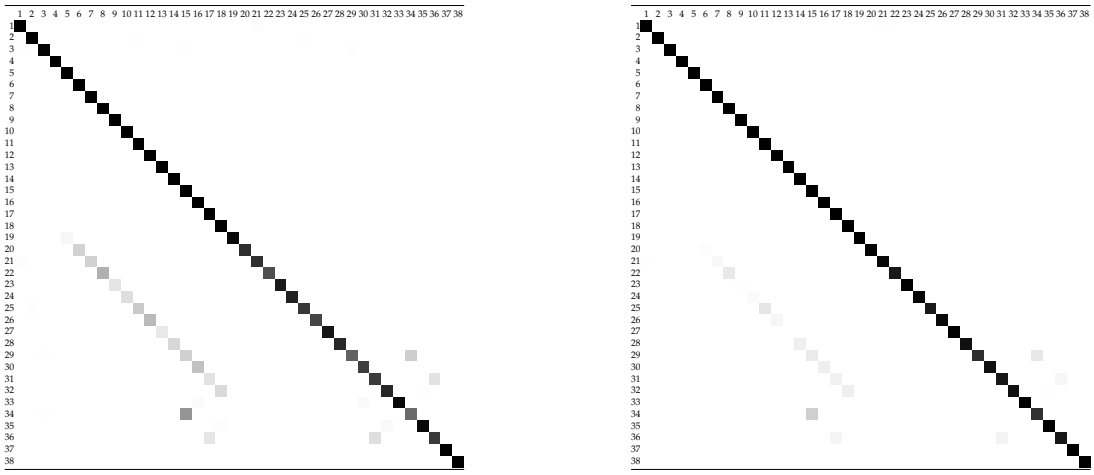


FIGURE 3.5: Confusion matrices: Probability with which a password of one class (row) is classified as another class (column) by $\mathcal{H}_k$ (Fig. 3.5a) or $\mathcal{B}$ with $|U| > 1$ (Fig. 3.5b). Box shading is scaled linearly between 0.0 (white) and 1.0 (black).

LastPass with "U", "L", "S", and "N" checked ($\hat{i}_* = 32$).

Using these online generators, we generated three datasets, namely a training dataset, an evaluation dataset, and a test dataset, all consisting of passwords of length 14 only. We used the training dataset to train a classifier to classify random passwords and evaluated it on the evaluation dataset. To assemble the training dataset and evaluation dataset, we generated 8,000 and 2,000 passwords from each class, yielding 304,000 and 76,000 passwords in total, respectively. We applied test dataset in the evaluation of FPP and FNP. In the test dataset, there were 38 classes of passwords, each containing 10,000 sets (corresponding to 10,000 users) with 100 passwords of that class. When evaluating FPP and FNP, we implemented gen by sampling x and U without replacement from a set (user) chosen uniformly at random from the test dataset.

3.3.3.2 Experimental Results

Using exiting honeyword-generation methods to generate honeywords: We first show experimental results for existing honeyword-generation methods described in Sec. 3.1 in the case when passwords are algorithmically-generated. We report $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ on those

Table 3.3: Classes of algorithmically generated passwords used in our experiments

Class index	Manager	Type	Alphabet			
			U	L	S	N
1	LastPass	Easy to say		✔		
2	LastPass	Easy to say	✔			
3	LastPass	Easy to say	✔	✔		
4	LastPass	Easy to read			✔	
5	LastPass	Easy to read				✔
6	LastPass	Easy to read			✔	✔
7	LastPass	Easy to read		✔		
8	LastPass	Easy to read		✔	✔	
9	LastPass	Easy to read		✔		✔
10	LastPass	Easy to read		✔	✔	✔
11	LastPass	Easy to read	✔			
12	LastPass	Easy to read	✔		✔	
13	LastPass	Easy to read	✔			✔
14	LastPass	Easy to read	✔		✔	✔
15	LastPass	Easy to read	✔	✔		
16	LastPass	Easy to read	✔	✔	✔	
17	LastPass	Easy to read	✔	✔		✔
18	LastPass	Easy to read	✔	✔	✔	✔
19	LastPass	All characters				✔
20	LastPass	All characters			✔	✔
21	LastPass	All characters		✔		
22	LastPass	All characters		✔	✔	
23	LastPass	All characters		✔		✔
24	LastPass	All characters		✔	✔	✔
25	LastPass	All characters	✔			
26	LastPass	All characters	✔		✔	
27	LastPass	All characters	✔			✔
28	LastPass	All characters	✔		✔	✔
29	LastPass	All characters	✔	✔		
30	LastPass	All characters	✔	✔	✔	
31	LastPass	All characters	✔	✔		✔
32	LastPass	All characters	✔	✔	✔	✔
33	1Password	Random Password	✔	✔	✔	
34	1Password	Random Password	✔	✔		
35	1Password	Random Password	✔	✔	✔	✔
36	1Password	Random Password	✔	✔		✔
37	1Password	Memorable Password		✔		✔
38	1Password	Memorable Password		✔	✔	✔

honeyword-generation methods for the attackers $\mathcal{F}$ and $\mathcal{B}$ described in Sec. 3.3.1. To depict the tradeoffs between $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$, we plot them against one another as τ is varied. Fig. 3.6 shows these tradeoffs when $k = 19$ honeywords and $\gamma = 1000$, where circles ($\bullet$) mark the $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ tradeoff at specific values of τ ranging from $\tau = 1$ to k in each plot. Again, we stress that $\gamma = 1000$ yields an optimistic evaluation of $\mathsf{FPP}(\mathcal{F})$ since $\gamma = 1000$ is $1000\times$ too small compared with the number 10^6 recommended by Florêncio et al. [46].

As seen in Fig. 3.6, the best $\mathsf{FNP}(\mathcal{B})$ that the honeyword-generation techniques accomplish is 0.27 (cbt-3, Fig. 3.6e), 0.37 (cbt-4, Fig. 3.6j), 0.55 (cbt*, Fig. 3.6k), and 0.59 (ptep, Fig. 3.6o). All others suffer from $\mathsf{FNP}(\mathcal{B}) > 0.79$, and input-independent methods including list, mkv, pcfg, rnn, and cbm have $\mathsf{FNP}(\mathcal{B}) \approx 1$. Intuitively, those methods, except cbt, generate honeywords based on a pretrained password generative models. Most of their generated honeywords do not fit the pattern of algorithmically generated passwords, and thus it is easy for $\mathcal{B}$ to distinguish the account password from the honeywords. In contrast, cbt generates honeywords by random replacement. Although random replacement still breaks the pattern from an algorithmic password generator with some probability, it produces honeywords looking more like algorithmically generated passwords, making it challenging for $\mathcal{B}$ to distinguish them. In the evaluation of $\mathsf{FPP}(\mathcal{F})$, none of the honeyword-generation methods achieved $\mathsf{FPP}(\mathcal{F}) \approx 0$ at $\tau = 1$. cbt-3, which accomplished the best $\mathsf{FNP}(\mathcal{B})$, has a high $\mathsf{FPP}(\mathcal{F}) = 0.37$. The best method in $\mathsf{FPP}(\mathcal{F})$ is cbt-4, which achieves 0.09, still much larger than 0. Growing τ reduces $\mathsf{FPP}(\mathcal{F})$ but increases $\mathsf{FNP}(\mathcal{B})$. Those methods that achieve $\mathsf{FPP}(\mathcal{F}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.80$.

A natural method to attempt to decrease $\mathsf{FNP}(\mathcal{B})$ would be to increase the number k of honeywords, but the more pronounced effect of doing so is increasing $\mathsf{FPP}(\mathcal{F})$, instead. Indeed, Fig. 3.7 shows the impact of increasing k to $k = 99$. As seen there, an order-of-magnitude increase in k resulted in a slight improvement to $\mathsf{FNP}(\mathcal{B})$ in each case, but a more substantial increase to $\mathsf{FPP}(\mathcal{F})$. All honeyword-generation techniques described in Sec. 3.1 failed to achieve low $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ simultaneously when the user employs

an algorithmic password generator.

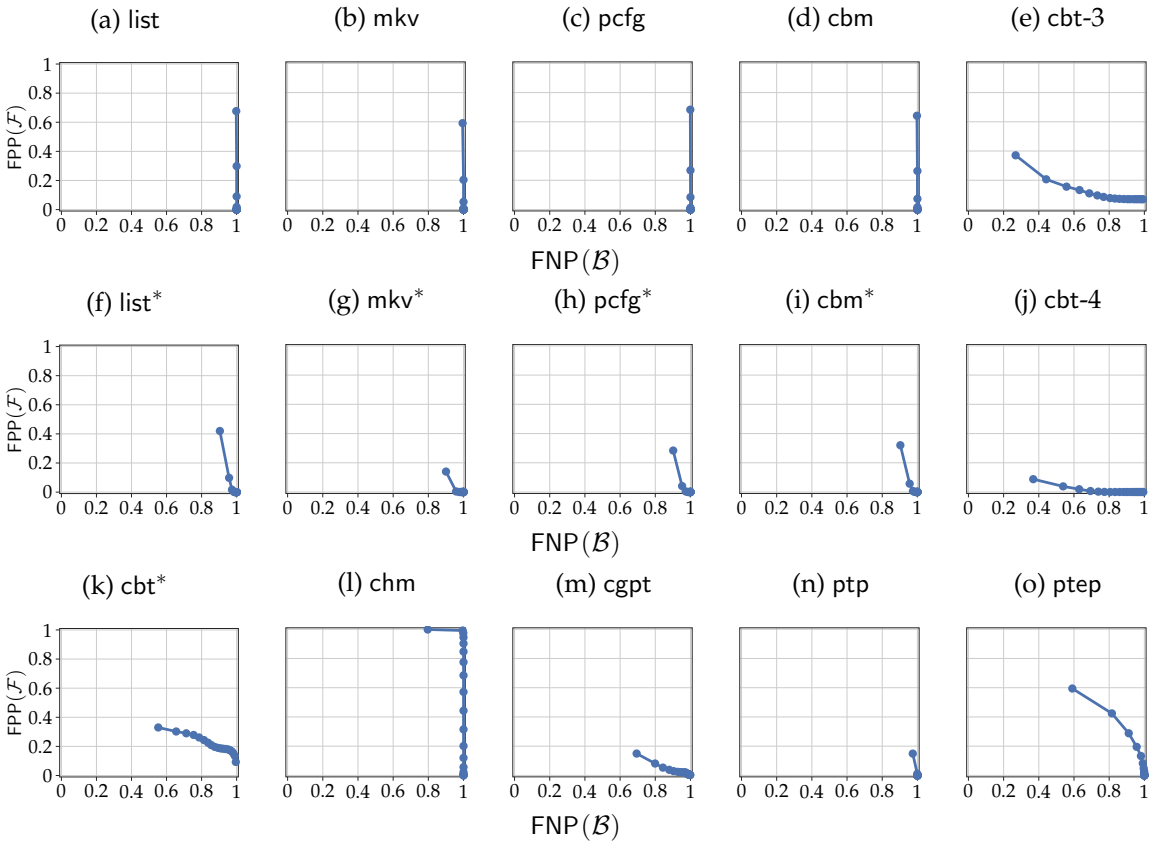


FIGURE 3.6: $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ as τ is varied, for algorithmically generated passwords ($k = 19$, $\gamma = 1000$). The best $\mathsf{FNP}(\mathcal{B})$ are 0.27 (cbt-3, Fig. 3.6e), 0.37 (cbt-4, Fig. 3.6j), 0.55 (cbt*, Fig. 3.6k), and 0.59 (ptep, Fig. 3.6o); all others have $\mathsf{FNP}(\mathcal{B}) > 0.69$. All suffer $\mathsf{FPP}(\mathcal{F}) > 0.09$ at $\tau = 1$. Those that reach $\mathsf{FPP}(\mathcal{F}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.80$.

Using algorithmic password generators to generate honeywords: We evaluated the honeyword-generation methods described in Sec. 3.3.2, though we plot only $\mathsf{FNP}(\mathcal{B})$ since $\mathsf{FPP}(\mathcal{F})$ was essentially perfect. We plot $\mathsf{FNP}(\mathcal{B})$ against τ in Fig. 3.8. As seen there, both the fxd and rgm methods had a high $\mathsf{FNP}(\mathcal{B})$. Even when $\tau = 1$, they had $\mathsf{FNP}(\mathcal{B}) > 0.94$ for $k = 19$ and $\mathsf{FNP}(\mathcal{B}) > 0.93$ for $k = 99$.

In contrast, the cls method achieves nearly perfect $\mathsf{FNP}(\mathcal{B})$. This method selects the most plausible algorithmic password generator based on the account password to generate honeywords. The confusion matrix experienced by $\mathcal{H}_k$ (i.e., using x) are shown in Fig. 3.5a.

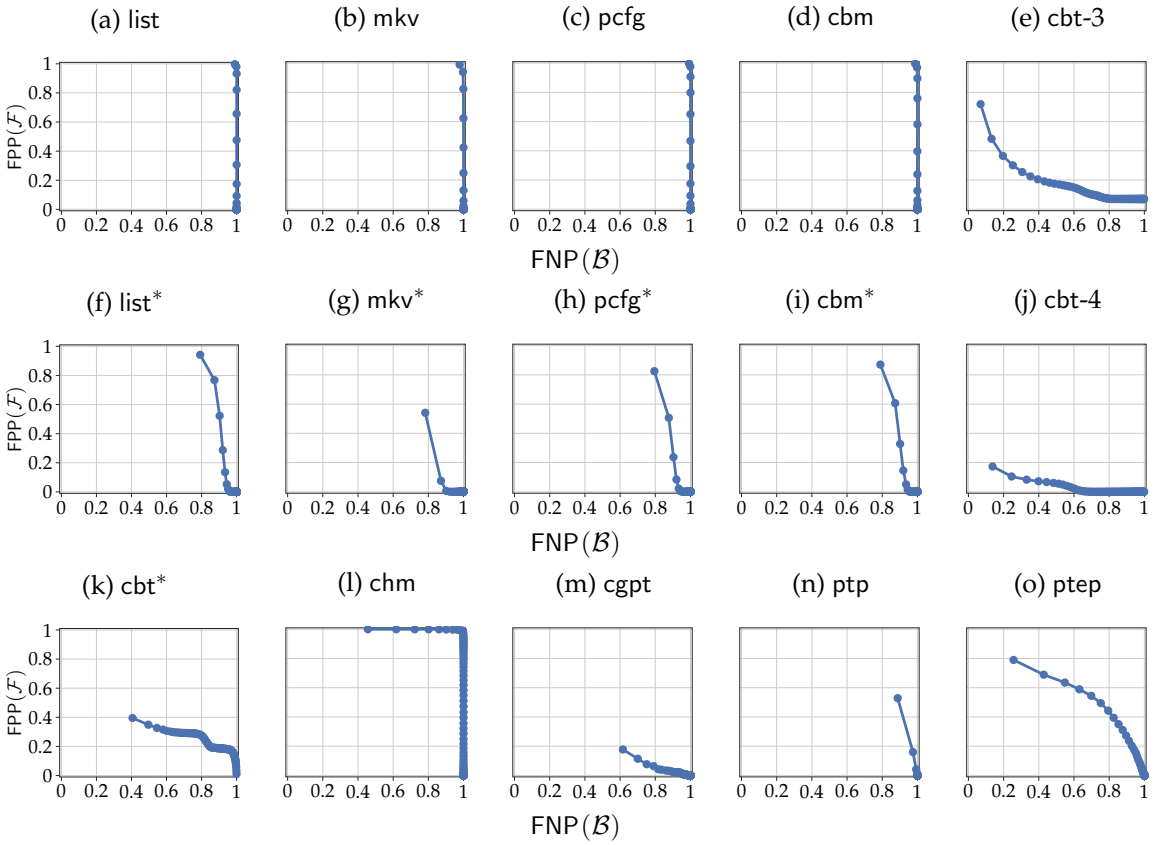


FIGURE 3.7: $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ as τ is varied, for algorithmically generated passwords ($k = 99$, $\gamma = 1000$). The best $\mathsf{FNP}(\mathcal{B})$ are 0.07 (cbt-3, Fig. 3.7e), 0.14 (cbt-4, Fig. 3.7j), and 0.25 (ptep, Fig. 3.7o); all others have $\mathsf{FNP}(\mathcal{B}) > 0.40$. All suffer $\mathsf{FPP}(\mathcal{F}) > 0.17$ at $\tau = 1$. Those that reach $\mathsf{FPP}(\mathcal{F}) \approx 0$ do so with $\mathsf{FNP}(\mathcal{B}) > 0.70$.

When $|U| = 1$, the confusion experienced by $\mathcal{B}$ is virtually identical, of course, but the confusion experienced by $\mathcal{B}$ when $|U| > 1$ is notably less, as shown in Fig. 3.5b. As this figure shows, when $|U| > 1$, $\mathcal{B}$ has greater ability to classify the user's password generator based on U than $\mathcal{H}_k$ does based on x, at least for certain classes. Since our dataset is dominated by accounts for which the number of passwords known by $\mathcal{B}$ numbers $|U| = 1$ (Fig. 3.2), the confusion shown in Fig. 3.5a (where $|U| > 1$) cannot effectively be exploited by $\mathcal{B}$.

However, if the fraction of accounts for which $\mathcal{B}$ holds $|U| > 1$ passwords were larger, the better classification accuracy this would enable (Fig. 3.5b) would permit an average increase

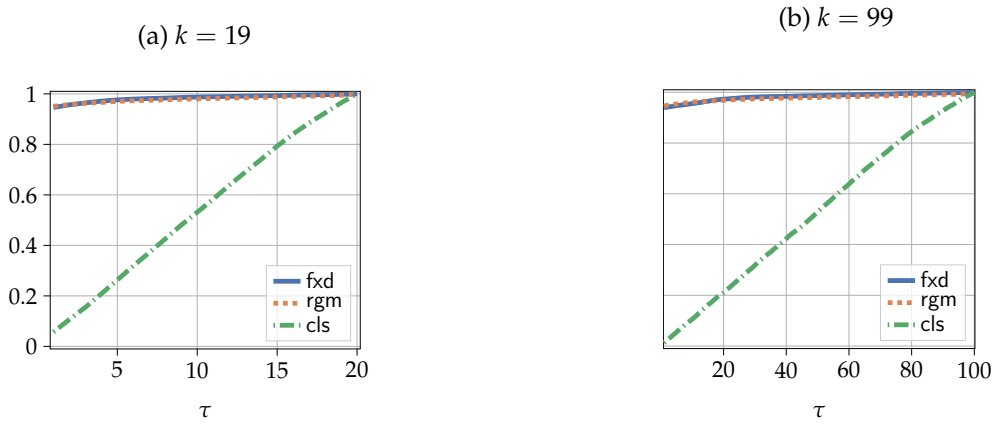


FIGURE 3.8: $\mathsf{FNP}(\mathcal{B})$ for honeywords by algorithmic password generators

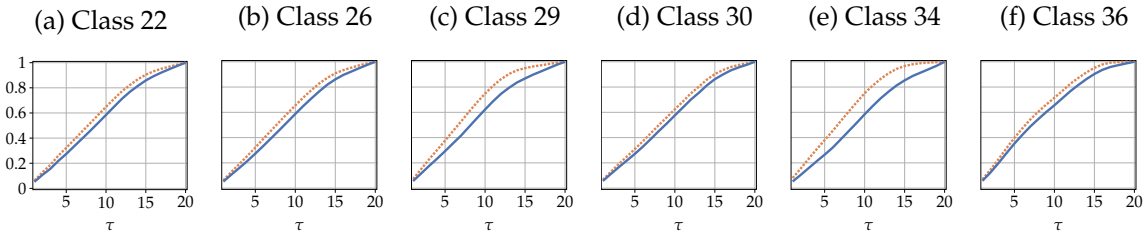


FIGURE 3.9: $\mathsf{FNP}(\mathcal{B})$ for $\mathcal{H}_k$ of cls ($k = 19$), for $|U|$ distributed as in Fig. 3.2 (–) and for $|U| = 99$ (--)

in $\mathsf{FNP}(\mathcal{B})$. To illustrate this, in Fig. 3.9 and Fig. 3.10 we show the effect on $\mathsf{FNP}(\mathcal{B})$ of increasing $|U|$ from its original distribution to $|U| = 99$ always, for $k = 19$ or $k = 99$, respectively. Each subfigure shows $\mathsf{FNP}(\mathcal{B})$ for certain classes of the actual password x; e.g., Fig. 3.9a shows this effect when $x \leftarrow \mathsf{gen}_{22}()$. As can be seen in these figures, increasing $|U|$ to $|U| = 99$ enables $\mathcal{B}$ to improve $\mathsf{FNP}(\mathcal{B})$, for these classes noticeably.

In conclusion, in the case of algorithmically generated passwords, it is critical for $\mathcal{H}_k$ to identify the algorithmic password generator used by each user in order to achieve low $\mathsf{FNP}(\mathcal{B})$. Even then, as the number of passwords $|U|$ grows, this measure will decay.

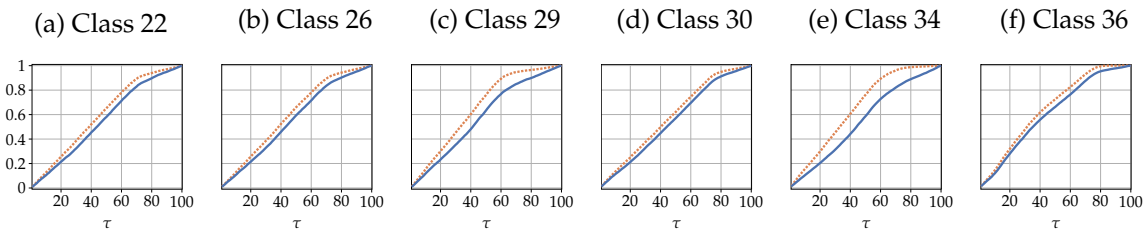


FIGURE 3.10: $\mathsf{FNP}(\mathcal{B})$ for $\mathcal{H}_k$ of cls ($k = 99$), for $|U|$ distributed as in Fig. 3.2 (–) and for $|U| = 99$ (--)

3.4 Discussion

3.4.1 False-Negative Attacks with Less Auxiliary Information

In Sec. 3.2, we assumed that the false-negative attacker has knowledge of the passwords used by the same user at other sites. We additionally explored a setting where the false-negative attacker had minimal auxiliary information about the users, specifically only one password used by the same user at another site ($|U| = 1$). To do so, we implemented the algorithm gen by choosing two passwords without replacement from a single multiset chosen uniformly at random from 4iQ test sets, returning one as x and the other as the only element of U. The experimental results in this setting are shown in Fig. 3.11. When the false-negative attacker $\mathcal{B}$ had little information about users, his success rate of guessing the user passwords from sweetwords slightly dropped, yielding a smaller $\mathsf{FNP}(\mathcal{B})$. However, all the honeyword-generation methods remained insufficiently resilient to these attacks. The best $\mathsf{FNP}(\mathcal{B})$ that the honeyword-generation techniques accomplished for all accounts was 0.48 (ptep, Fig. 3.11r). No existing algorithm achieved low rates of both false positives and false negatives.

3.4.2 Countermeasures to False-Positive Attacks

False-positive attacks can be very costly to sites, if they induce an investigation into the possibility of a breach and/or a password reset for every site account. Moreover, repeated false positives might eventually result in the defense being ignored or disabled outright. Despite the consequences of false positives, only a few previous works on honeywords have

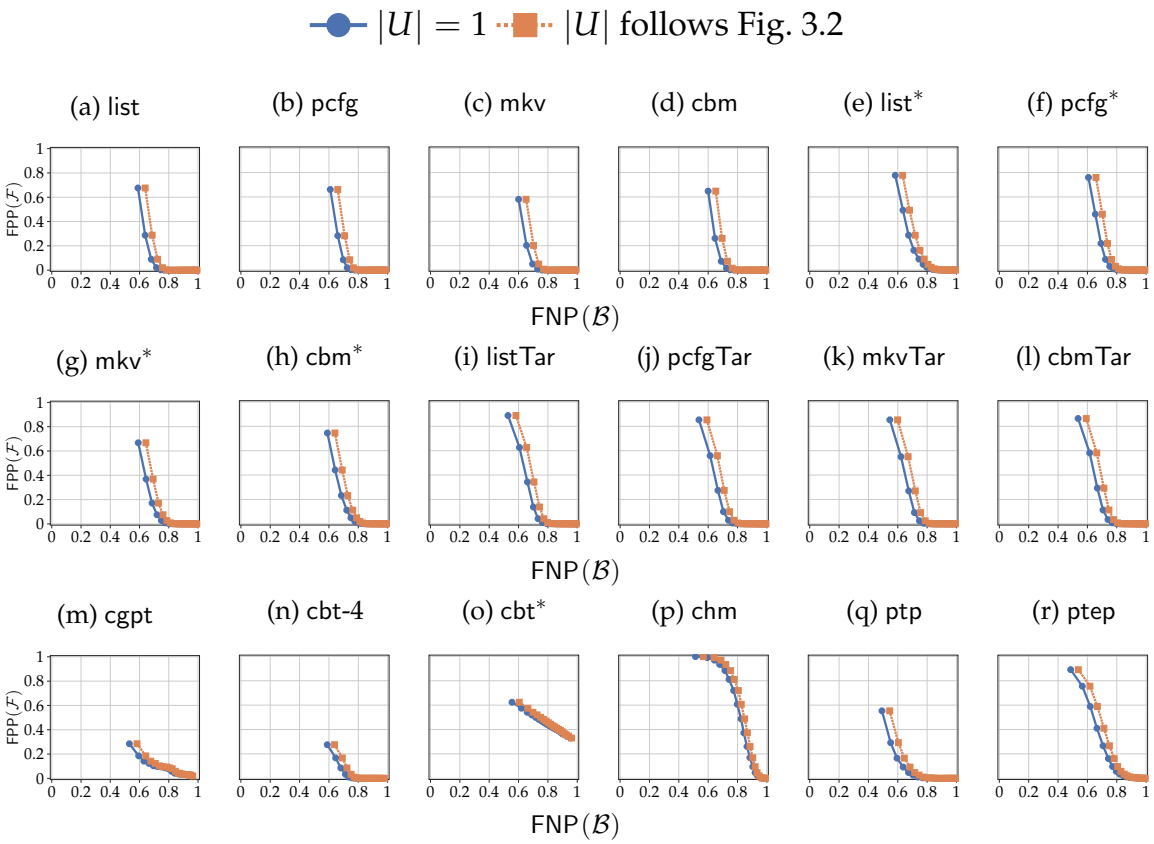


FIGURE 3.11: FPP($\mathcal{F}$) vs. FNP($\mathcal{B}$) as $|U|$ is varied, for the case of user-chosen passwords ($k = 19$, $\gamma = 1000$) and all accounts.

briefly discussed how to prevent them [69, 157]. Wang et al. [157] suggested that applying a blocklist of popular passwords to honeyword selection can reduce false positives, since the honeyword-generation methods considered in their work generate honeywords by sampling from a public password distribution (e.g., leveraging a password model like list). As such, a blocklist would avoid using popular passwords as honeywords, which can mitigate the guessability of honeywords by their proposed methods. However, a blocklist of popular passwords is much less effective when considering input-dependent honeyword-generation algorithms (e.g., cbt, chm, ptp, and ptep), since these methods assign more likelihood to those candidates similar to the user password. A way to mitigate false positives of these methods is to avoid using passwords similar to the user password as honeywords, which makes them suffer a high false-negative rate.

Another countermeasure to reduce false positives, as mentioned by Juels and Rivest [69], is to select k honeywords uniformly at random from a large pool of candidate honeywords that are similar to the user password. In order to achieve a small false-positive rate, the size of the pool should be much larger than k. However, it is challenging to generate such a large pool of candidates that are sufficiently similar to the user password to ensure a small false-negative rate via this process. As such, an interesting direction is to explore how to generate such a large candidate pool to achieve a target false-negative rate.

Ultimately, a site might find it most palatable to address the risk of false positives by adopting a lenient policy toward honeyword-induced breach alarms. Previous works (e.g., [69, Sec. 2.4]) outlined a range of possible reactions to honeyword-induced alarms, ranging from severe (e.g., shutting down the system and forcing all users to reset their passwords) to lenient (e.g., allow the login to proceed as usual). Whatever the policy, however, it will apply to both false and true positives alike, and so a policy can be relaxed only so far as is acceptable when a breach actually occurs.

3.4.3 Balancing Attention to False Positives and False Negatives

Since honeywords' proposal, a challenge has been to design good honeyword-generation methods that achieve both low false-positives and low false-negatives, i.e., $\mathsf{FPP} \approx 0$ and $\mathsf{FNP} = \frac{\tau}{k+1}$. However, our experimental results in Sec. 3.2.3.2 show that no existing method achieves this goal in a threat model in which passwords from the same user at other sites are exposed to the attacker. While this leaves us skeptical that a perfect honeyword-generation method exists for this threat model (at least not when passwords are user-chosen, versus algorithmically generated), we do not mean to suggest that research in this direction should end. However, we do advocate that new honeyword-generation methods should be investigated with balanced attention to false positives and false negatives in this threat model, rather than more narrowly focusing on false negatives, as has been typical in most prior research.

In the absence of improved honeyword-generation algorithms that more effectively bal-

ance false positives and false negatives, we believe that a change in perspective on the use of honeywords might be warranted. One such perspective, reflected in recent work [160], is to tightly constrain false positives to enable a honeyword-induced alarm to confidently be taken seriously and dealt with severely, and then to rely on a false-negative attacker's interests in harvesting *many* accounts to likely trigger a breach alarm even if the true-positive probability *per account* is modest. Such an approach will likely permit breaching attackers to harvest selected accounts without triggering a breach alarm, but greedy breaching attackers will still trigger a breach alarm with high probability. Based on these insights, we will propose LeakSentinel in the next chapter, which allows a site to detect a credential-database breach when the breaching attacker exploits a sufficient number of accounts, while maintaining a small, provably bounded global false-positive rate.

3.4.4 Assumptions on Algorithmic Password Generator Configuration

A limitation of our analysis in Sec. 3.3 is that it was conducted assuming that the user uniformly randomly selects a configuration for her algorithmic password generator and, once adopting a configuration, does not change it.

Changes of password manager configuration: To justify the assumption that users rarely change password manager configurations, we performed a study of the password-creation policies of twenty commonly accessed websites and those from Tranco Top 1M websites [113] to show that users are rarely *required* to change configurations. Specifically, we sought to determine the frequency with which a user who sets passwords at these sites in a random order will be required to change his password-creation algorithm configuration to comply with the next site in the sequence.

To perform this evaluation, we retrieved the password requirements from twenty commonly visited websites [129], shown in Table 3.4, or simulated password composition policies based on the statistics from a recent large-scale study [6], shown in Table 3.5. For a sequence of password policies that is a permutation of password requirements from the twenty commonly visited websites or a sequence of 101 simulated password-composition

Table 3.4: Most visited websites [129] and their password composition policies retrieved in May 2023.

Site	Password composition policy
google.com	$\geq$ 8 characters including a letter, a symbol, and a number
youtube.com	$\geq$ 8 characters including a letter, a symbol, and a number
facebook.com	$\geq$ 6 characters including a letter, a symbol, and a number
twitter.com	$\geq$ 8 characters including a letter, a symbol, and a number
instagram.com	$\geq$ 6 characters including a letter, a symbol, and a number
baidu.com	$\geq$ 8 and $\leq$ 14 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
wikipedia.org	$\geq$ 8 characters
yandex.ru	$\geq$ 6 characters including an uppercase letter, a lowercase letter, and a number
yahoo.com	$\geq$ 9 characters
xvideos.com	No requirement
pornhub.com	$\geq$ 6 characters
amazon.com	$\geq$ 6 characters
tiktok.com	$\geq$ 8 and $\leq$ 20 characters including a letter, a symbol, and a number
live.com	$\geq$ 8 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
openai.com	$\geq$ 8 characters
reddit.com	$\geq$ 8 characters
linkedin.com	$\geq$ 6 characters
netflix.com	$\geq$ 6 characters
office.com	$\geq$ 8 characters of at least two types from uppercase letter, lowercase letter, symbol, and number
twitch.tv	$\geq$ 8 characters

Table 3.5: Percentages of sites from Tranco Top 10K, 100K, and 1M that do not require some types of characters (U: uppercase letter, L: lowercase letter, S: special symbol, N: number). These statistics were obtained on Dec. 2021 [6].

Tranco Top	U	L	S	N
10K	81.7%	78.4%	80.3%	71.2%
100K	79.4%	79.2%	76.3%	82.5%
1M	83.7%	84.1%	86.3%	82.0%

Table 3.6: Evaluation results on random walking at websites.

Sites	Number of conflicts	Probability of conflict with the next site	Average number of non-conflicting sites before a conflict
20 commonly visited websites	2.143	0.1127	8.864
101 simulated sites from Tranco Top 10K	15.829	0.1529	6.317
101 simulated sites from Tranco Top 100K	14.450	0.1445	6.920
101 simulated sites from Tranco Top 1M	11.825	0.1182	8.456

policies randomly constructed from the statistics, we evaluated the number of times that the current password-generator configuration conflicted with the password-creation policy of the next website in the sequence, starting from a configuration initialized by the minimum password requirement of the first site. To ensure a conservative evaluation, when a conflict occurred, the current configuration was replaced with the minimum password requirements of the conflicting site. For each type of password policy sequence, we performed this analysis for 10^6 times. The evaluation results are shown in Table 3.6. We found that the numbers of conflicts ranged from an average of 2.143 in sequences of 20 websites to 15.829 in sequences of 101 simulated sites drawn from Tranco Top 10,000 sites, averaged over the 10^6 sequences. These implied a probability of conflict with the next site in the sequence between 0.1127 and 0.1529. In expectation, then, the probability of exactly κ consecutive resets with no conflicts, followed by a site that conflicts, is $< (1 - 0.1529)^{\kappa}(0.1529)$, and the average number of non-conflicting sites before a conflict was > 6.3. Given the conservative nature of our evaluation, we believe this result justifies our assumption that a user would rarely change its password-generator configuration.

However, an interesting direction for future work would be to confirm or refute this assumption more broadly, since as shown in Sec. 3.3.3.2, the assumption somewhat diminishes the effectiveness of honeywords generated for accounts with algorithmically generated passwords. Alternatively, an algorithmic password generator could be designed to encourage changing these configuration settings regularly, in which case an interesting research direction would be to explore the acceptability of this practice for users.

Tendency to use default password configuration: In reality, we expect people to generally

defer to the default password-manager configuration until forced to change it by a site's password policy. However, imposing a non-uniform distribution on $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$ to reflect this tendency should not qualitatively change the results of our study: First, analogous to our results in Sec. 3.3.3.2, the existing honeyword-generation methods would still fail to provide a low false-negative probability since the false-negative attacker would even more easily distinguish the algorithmically generated password from the honeywords if he knows how the selection of configuration is biased. Second, as shown in Sec. 3.3.3.2, when the honeyword system can correctly predict the configuration used by the user—which should only become easier when the distribution is biased toward the default—and use that configuration to generate the honeyword, the generated honeywords can provide sufficient security.

3.4.5 A Mixed Case Study

In this chapter, we studied two representative cases where users create user-chosen passwords (Sec. 3.2) or where users generate their passwords algorithmically using a password manager (Sec. 3.3). To assess the efficacy of honeywords when users employ mixed strategies (i.e., chose some passwords themselves and algorithmically generate others), we further constructed two test datasets by mixing 4iQ dataset and the algorithmically generated dataset. Then we generated honeywords based on the type of the user password, i.e., applying honeyword-generation methods described in Sec. 3.1.2 to generate honeywords for user-chosen passwords and password managers to generate honeywords for algorithmically generated passwords. Our study showed that increased use of password managers in password creation can ease the tensions brought on by password reuse and thus make better trade-offs between false-positive and false-negative rates of honeywords.

Mixed datasets construction: We constructed two test datasets by mixing up 4iQ dataset and the algorithmically generated dataset as follows: for a multiset from the processed 4iQ test dataset (described in Sec. 3.2.3.1), we uniformly at random sampled a password generator from $\{\mathsf{gen}_{\hat{i}}\}_{\hat{i}=1}^{\hat{n}}$ and randomly replace each password in the multiset by a password generated from sampled password generator independently with a probability of π. We chose π to be

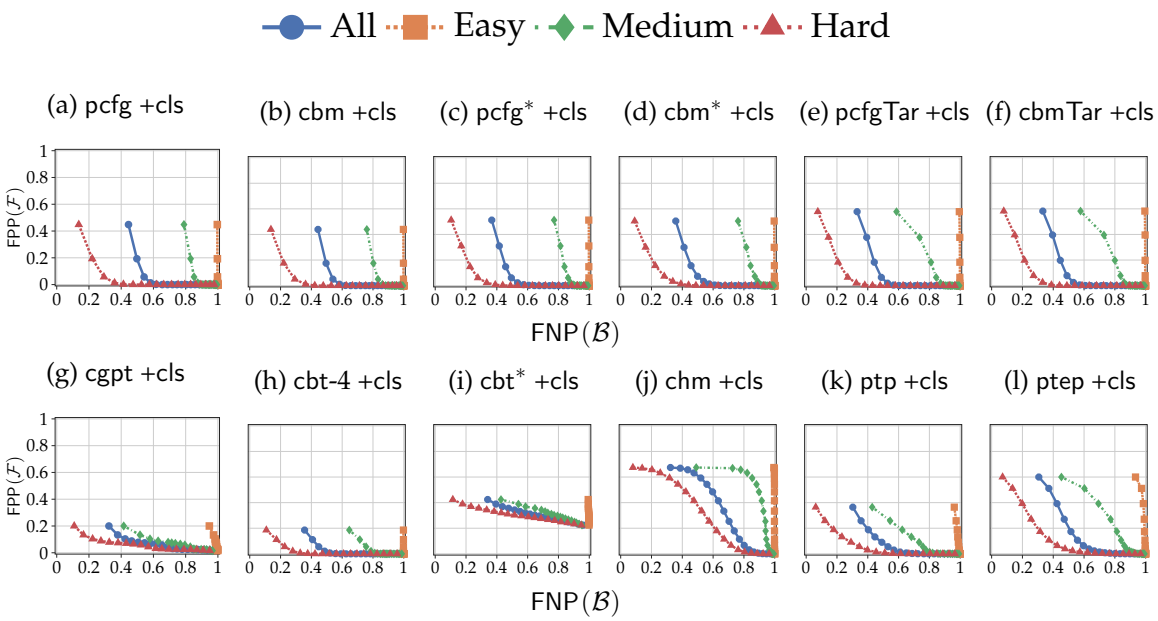


FIGURE 3.12: FPP($\mathcal{F}$) vs. FNP($\mathcal{B}$) as τ is varied, for the case of mixed passwords ($\pi = 0.33$) ($k = 19$, $\gamma = 1000$).

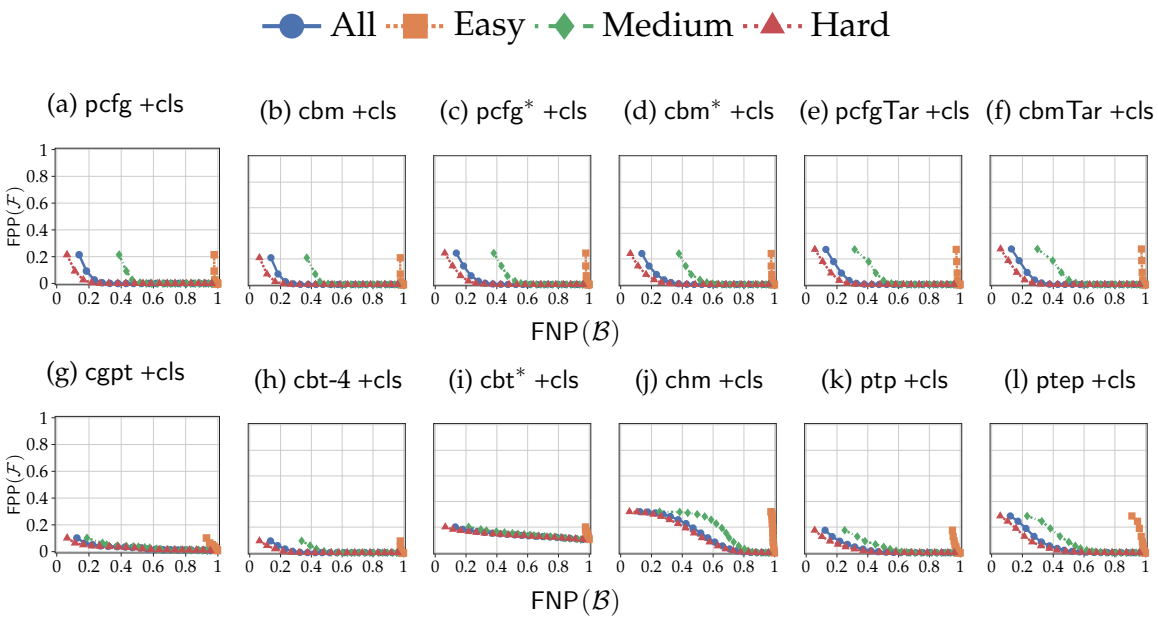


FIGURE 3.13: FPP($\mathcal{F}$) vs. FNP($\mathcal{B}$) as τ is varied, for the case of mixed passwords ($\pi = 0.67$) ($k = 19$, $\gamma = 1000$).

0.33 and 0.67. As such, we constructed two mixed datasets. To evaluate false-positive and false-negative of the honeyword methods, we implemented the algorithm gen by choosing x and the members of U without replacement from a single multiset, chosen uniformly at random from the mixed test datasets.

Honeyword generation: Given an user password x from the mixed test datasets, we generated honeywords based on the following rule: if x is an algorithmically generated password, we generated honeywords by cls (described in Sec. 3.3.1); if x is a user-chosen one, we generated honeywords by a method described in Sec. 3.1.2. Here we assumed that user-chosen passwords and algorithmically generated passwords can be distinguished easily.

Attack strategies: We implemented the false-positive attacker following the one described in Sec. 3.2.1. We implemented the false-negative attacker as follows: given the sweetwords $H \cup \{x\}$, if most of sweetwords are algorithmically generated, the false-negative attacker followed the strategy described in Sec. 3.3.1 to guess the account password; if most of sweetwords are user-chosen, he followed the strategies described in Sec. 3.2.1.

Experimental results: We report $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ on those honeyword-generation algorithms for the attackers $\mathcal{F}$ and $\mathcal{B}$ described above. To depict the tradeoffs between $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$, we plot them against one another as τ is varied. Fig. 3.12 (Fig. 3.13) shows these tradeoffs when $k = 19$ honeywords, $\gamma = 1000$, and $\pi = 0.33$ ($\pi = 0.67$) where circles ($\bullet$) mark the $\mathsf{FPP}(\mathcal{F})$ vs. $\mathsf{FNP}(\mathcal{B})$ tradeoff at specific values of τ ranging from $\tau = 1$ to k. Again, we stress that $\gamma = 1000$ yields an optimistic evaluation of $\mathsf{FPP}(\mathcal{F})$ since $\gamma = 1000$ is $1000\times$ too small compared with the number 10^6 recommended by Florêncio et al. [46]. Compared with those results from user-chosen case (depicted in Fig. 3.3), honeyword-generation methods achieved better trade-offs between $\mathsf{FPP}(\mathcal{F})$ and $\mathsf{FNP}(\mathcal{B})$ in the two mixed datasets. When we set π to be higher, we have better trade-offs because a higher π makes less password reuse across sites. From these results, we conclude that honeyword efficacy benefits from the tendency of using password manager in password creation.

3.4.6 Password Reuse

Our findings that password reuse across sites is so detrimental to honeyword false-negative rates (Sec. 3.2.3.2) provides yet more evidence that moving more users toward password managers would be good policy (notwithstanding the risk of password-manager breaches, e.g., [145]). That said, a recent university survey [95] found that though a large majority (77%) of respondents reported using a password manager, another large majority (again, 77%) also reported still reusing passwords across accounts. So, while a step in the right direction, password managers are evidently not a panacea. A potentially more effective approach might be explicitly hindering attempts to reuse passwords, either through adoption of intentionally conflicting password requirements at websites (which is not commonplace, see Sec. 3.4.4) or through explicit interventions during the password (re)setting process to interfere with reusing the same or similar passwords (e.g., [161]).

3.5 Chapter Summary

In this chapter, we have conducted the first critical analysis of honeyword-generation techniques for users who have suffered exposed passwords for their accounts at other sites. We formalized the false-positive rate and false-negative rate of honeywords in a model where the attacker has access to passwords for the same users at other sites or, in the case of false-positive attackers, even passwords for users at the defending site (as the real users would). Using these formalized definitions and a large dataset of leaked passwords, we experimentally demonstrated that existing honeyword-generation algorithms exhibit poor tradeoffs between false positives and false negatives when the account password is chosen by an average human user. Then we studied the case where the account password is algorithmically generated and used passwords from popular password managers to show that the existing honeyword-generation methods offer modest protection against false-negative attackers. We further explored the use of algorithmic password generators in honeyword generation and determined that seemingly the only effective strategy is to generate honeywords using the same password generator that the user does, if it can determine what that password generator is. In total,

we believe our results paint a cautionary picture for the state of honeyword-generation algorithms to date, though they also set forth new research challenges for the field.

4. LeakSentinel: A Honeyword Framework for Anytime-Valid Credential-Database Breach Detection

In this chapter, we propose a new honeyword framework for detecting credential-database breaches. Honeyword-based detection relies on two central requirements: good flatness and small false-detection rates. However, achieving both good flatness and a small false-detection rate has proven elusive. Existing works [4, 44, 52, 157, 175] primarily focus on improving flatness by generating honeywords that are difficult to distinguish from user passwords. However, these approaches typically rely on a simple detection mechanism that raises a breach alarm whenever a sufficient number of honeywords are observed in login attempts. As a result, they suffer from a high and unbounded false-detection rate in the presence of false-alarm attackers. For example, when using a hybrid method proposed by Wang et al. [157] that combines a list-based model, a Markov model, and a probabilistic context-free grammar (PCFG) model to generate 19 honeywords per account, a false-alarm attacker can trigger a false alarm with probability up to 20% by only 10^3 login attempts, even after the site used a blocklist with 10^5 passwords to mitigate it [157, Fig. 4]. In Chapter 3, we evaluated 22 honeyword-generation methods and showed that, when 19 honeywords were generated per account, these methods exhibited a false-detection rate of at least 27% under 10^3 login attempts by a false-alarm attacker.

To our knowledge, Bernoulli Honeywords [160] is the first method to provide an analytic false-detection rate *per online-guessing campaign*. For example, considering a breadth-first false-alarm attacker who launches 10^4 login attempts per account against 10^3 accounts, or a depth-first false-alarm attacker who tries 10^6 login attempts per account against 10 accounts, their bounds ensure that such a false-alarm attack campaign triggers a breach alarm with probability less than 10%. Assuming only one false-alarm attack campaign happens in every 4 months, this corresponds to less than one false detection every 3 years in expectation [160, p. 5]. Still, however, this method lacks a bound on the false-detection rate *independent of any assumptions on the attack intensity*—that is, assumptions about how aggressively a false-alarm attacker issues login attempts over time and across accounts

(e.g., the number of login attempts, the number of targeted accounts, and the frequency of such attacks). Consequently, it remains an important yet largely unexplored problem to design a honeyword-based detection method that can effectively detect credential-database breaches while providing a tunable and provably bounded false-detection rate, independent of any such assumptions.

In this chapter, we propose LeakSentinel, a honeyword framework that allows a site to detect credential-database breaches. LeakSentinel is *anytime-valid* in that it guarantees a provably bounded global false-detection rate per site deployment, without requiring assumptions about attack intensity. It consists of a honeyword-generation algorithm and a breach-detection algorithm. The honeyword-generation algorithm applies any password generative model within either a weighted-sampling-without-replacement process or a Bernoulli-sampling process, thereby subsuming most existing honeyword-generation methods. Our breach-detection algorithm operates online and leverages a sequential hypothesis test to continuously analyze incorrect login attempts without a predetermined termination time. Our breach-detection algorithm permits the specification of a parameter $\alpha < 1$ so that, in the absence of credential-database breach, the probability that it ever raises an alarm over an infinite sequence of login attempts is provably bounded by α. LeakSentinel enables effective detection of credential-database breaches when a sufficient number of accounts are exploited by a breach attacker. We empirically validate the false-detection guarantee and the breach-detection effectiveness through extensive experiments. Specifically, when configured with $\alpha = 10^{-5}$ for a site with 10^7 users, LeakSentinel was robust to false-alarm attacks and, across the password generative models we evaluated, successfully detected all considered breach attacks once the attacker exploited a sufficient fraction of accounts, ranging from a single account to 43% of accounts. Moreover, its detection performance scaled favorably with the number of users, and it remained effective even at relatively small deployment scales.

To summarize, our contributions are as follows:

- We propose LeakSentinel, the first honeyword framework that enables effective de-

tection of credential-database breaches and provides a provably bounded global false-detection rate per site deployment. It comprises a randomized honeyword-generation algorithm that supports arbitrary password generative models and a novel breach-detection algorithm that operates online by leveraging a sequential test over incorrect login attempts.

- We empirically validate the false-detection guarantee and the breach-detection effectiveness through extensive experiments, spanning a wide range of attacker knowledge, capabilities, and strategies.

4.1 Preliminary

4.1.1 Honeyword-Based Authentication System

Consider a site with n registered users whose passwords are $x_1, x_2, \ldots, x_n$ respectively. We refer to these passwords $x_1, x_2, \ldots, x_n$ as *user passwords*, to distinguish them from honeywords introduced later. The user passwords are stored as salted hashes in a credential database. In a honeyword-based authentication system, the site augments the credential database with honeywords to detect the credential-database breaches. If an attacker compromises the credential database and uses the honeywords in login attempts, this would raise the site's attention that a breach has happened.

Setup: The site leverages a honeyword-generation algorithm to generate a set of honeywords $H_i = \{h_{i,1}, h_{i,2}, \ldots, h_{i,k_i}\}$ for each account $i \in \{1, 2, \ldots, n\}$. Then it constructs a sweetword set $S_i = \{x_i, h_{i,1}, \ldots, h_{i,k_i}\}$ per account, randomly shuffles the order of sweetwords in S_i, and stores their salted hashes in the credential database. In this chapter, we adopt the classical honeyword system design that leverages a separate component called honeychecker [69]. The honeychecker stores the index secret_i of the user password x_i in S_i, together with the user ID i. Subsequently, the site initializes a breach-detection algorithm that monitors the login attempts to detect whether a credential-database breach has happened.

Authentication: When there is a login attempt consisting of a user ID i and a password x', the system checks whether $x' \in S_i$ by comparing the salted hash of x' with those of

sweetwords in S_i. If $x' \notin S_i$, the system rejects the login attempt. If $x' \in S_i$, the system sends the index j^* of x' in S_i together with the user ID i to the honeychecker, which checks whether $j^* = \mathsf{secret}_i$. If $j^* = \mathsf{secret}_i$, the honeychecker returns a "success" to the system and then the system permits the login attempt; otherwise, the honeychecker returns a "fail" and then the system rejects the attempt. When $x' \notin S_i$ or honeychecker returns a "fail", the system sends this login attempt to the breach-detection algorithm, which analyzes and decides whether a credential-database breach has happened.

Honeyword deletion: In a honeyword-based system that incorporates LeakSentinel, we introduce an additional mechanism that removes honeywords upon use. Specifically, when a login attempt for account i uses a password $x' \in S_i$, the system sends the index j^* of x' in S_i to the honeychecker along with i. If the honeychecker determines that $j^* \neq \mathsf{secret}_i$, it returns a "fail" and updates $\mathsf{secret}_i \leftarrow \mathsf{secret}_i - 1$ whenever $j^* < \mathsf{secret}_i$ to maintain consistency of indices. Upon receiving the "fail" response, the system rejects the login attempt and removes x' from S_i, while preserving the relative order of the remaining sweetwords. Note that this mechanism won't delete the user password from the sweetword set. When the number of remaining passwords in S_i falls below a threshold (set to 1 by default), the system requires the user to reset the account, after which a new honeyword set is generated. This honeyword-deletion mechanism is essential for achieving our false-detection guarantees, as formalized in Sec. 4.3.2 and Sec. 4.3.3.

4.1.2 Password Generative Model

The honeyword-generation algorithm by a site uses a generative password model to generate honeywords [38, 61, 157]. A generative password model is a probabilistic model that assigns a probability to each password candidate from a password space $\mathcal{X}$ that includes all the possible passwords allowed by a site, e.g., passwords of length between 4 and 30. Such a probability distribution could be learned from a password dataset [90, 157, 167, 169] or heuristically constructed [69]. The distribution may be either independent of input or conditioned on information (e.g., user password or PII). Accordingly, generative password

models can be categorized as input-independent or input-dependent. We have introduced examples of password generative models in Sec. 3.1.

In this chapter, we consider input-independent models and those input-dependent models conditioned on user password. Some prior work (e.g., [157]) leveraged input-dependent password generative models that incorporated users' PII in honeyword generation. However, leveraging such information introduces the risk of leaking users' private information. For this reason, although our honeyword-generation algorithm (introduced in Sec. 4.3.1) can in principle be extended to incorporate auxiliary information, we intentionally refrain from doing so in our design. Without loss of generality, we define a generative password model $\mathcal{G}$ as one that, given user password x, assigns a probability $\mathcal{G}_x(x')$ to each password candidate $x' \in \mathcal{X}$, such that $\sum_{x' \in \mathcal{X}} \mathcal{G}_x(x') = 1$. For an input-independent model, we have $\mathcal{G}_{x_i}(x') = \mathcal{G}_{x_{i'}}(x')$ for any x_i, $x_{i'}$, and x'.

4.2 Threat Model and Design Goals

4.2.1 Threat Model

We consider two types of attackers against a honeyword-based authentication system: false-alarm attacker and breach attacker. We distinguish these two attackers based on their knowledge and goals.

False-alarm attacker: A false-alarm attacker does not compromise the credential database and therefore does not know which passwords are (not) selected as honeywords. We assume that the attacker knows a subset of $\{x_1, x_2, \ldots, x_n\}$ (e.g., by registering some accounts at the site), which is different from the false-positive attacker (introduced in Chapter 3) who knows only one account's user password. Consistent with prior work (e.g., [20]) and Chapter 3, we further assume that he knows the honeyword-generation algorithm and breach-detection algorithm used by the site. His goal is to trigger the site's breach alarm by guessing honeywords associated with user accounts and submitting them in login attempts.

Breach attacker: A breach attacker has compromised the credential database and therefore has access to the set of sweetwords for each account (i.e., he knows which passwords are not

selected as honeywords, distinguished from the false-alarm attacker). Consistent with prior work (e.g., [61, 156, 157]), we assume that the breach attacker can recover the plaintexts of the sweetwords from the salted hashes. Given the plaintext sweetwords, however, the breach attacker cannot reliably distinguish the user passwords from honeywords. He could utilize auxiliary knowledge (e.g., population-level password distribution [156, 157] or the passwords from the same users at other sites as considered in Chapter 3) to improve this distinction. In addition, we assume the breach attacker has the full knowledge of the honeyword-generation algorithm and the breach-detection algorithm. Different from the false-negative attacker (introduced in Chapter 3) who aims to exploit one account, the goal of the breach attacker is to exploit as many accounts as possible by identifying and submitting the user passwords from the sweetword sets in login attempts before alarming the system.

4.2.2 Design Goals

We aim to design a honeyword framework that a site can use to detect credential-database breaches. We have the following design goals for the framework:

- Tunable, provably bounded false-detection rate: This goal requires that when there is no credential-database breach, the probability that the framework ever raises an alarm over an infinite sequence of login attempts is provably bounded by a tunable constant. The provably bounded false-detection rate should hold under any false-alarm attack.
- Effectiveness of breach detection: This goal means that our framework enables a site with a sufficiently large user base (e.g., $n \geq 10^3$) to effectively detect credential-database breaches. Specifically, once a breach occurs and a breach attacker begins exploiting accounts using credentials from the breached database, the probability that our framework raises an alarm increases with the fraction of compromised accounts and approaches 100% as most accounts become compromised. This goal should hold even if the breach attacker knows how our approach works and accordingly applies an adaptive strategy to mitigate our detection.

4.3 Our Honeyword Framework: LeakSentinel

In this section, we introduce LeakSentinel, which enables a site to effectively detect credential-database breaches while providing a tunable and provably bounded global false-detection rate. LeakSentinel consists of a honeyword-generation algorithm and a breach-detection algorithm. The honeyword-generation algorithm is used to generate honeywords for accounts when they are newly registered or reset, while the breach-detection algorithm operates online to analyze the login attempts at login time. We introduce them in Sec. 4.3.1 and Sec. 4.3.2 respectively.

4.3.1 Honeyword Generation

We model the honeyword generation by a randomized algorithm $\mathcal{H}.$ that takes as inputs the user password x_i of a user i and outputs a set of honeywords:

$$H_i \leftarrow \mathcal{H}(x_i). \tag{4.1}$$

We consider two classes of honeyword-generation algorithms; both apply a password generative model $\mathcal{G}$. They are categorized into *weighted sampling without replacement* (WSWoR) and *Bernoulli sampling*.

4.3.1.1 WSWoR

This type of honeyword-generation algorithm generates a fixed, preselected number of honeywords per account (i.e., k_i is a constant across all users). It does so by selecting k_i password candidates using weighted sampling without replacement, where the selection weight of each password candidate is defined by a password generative model $\mathcal{G}$. Formally, for a user i with her user password x_i, given a password generative model $\mathcal{G}$ that assigns a probability $\mathcal{G}_{x_i}(x')$ to each password candidate $x' \in \mathcal{X}$, it generates the honeywords H_i by successive sampling: at each iteration, a candidate $x' \in \mathcal{X} \setminus \{x_i\}$ is selected as honeyword with probability proportional to $\mathcal{G}_{x_i}(x')$, conditioned on the password candidates not yet selected. As such, for each $x' \in \mathcal{X} \setminus \{x_i\}$, we have:

$$\mathbb{P}\big(x' \in H_i \mid x_i\big) = \sum_{\substack{X \subset \mathcal{X} \setminus \{x_i\} \\ \text{s.t. } x' \in X, |X| = k_i}} R(X, \mathcal{X} \setminus \{x_i\}, k_i), \tag{4.2}$$

where $R(X, V, m)$ denotes the probability that, when the password candidates not yet selected are $V \subseteq \mathcal{X} \setminus \{x_i\}$, drawing m more items ends up drawing exactly the items in $X \cap V$. We have the recursion:

$$R(X, V, m) = \sum_{x'' \in V} \frac{\mathcal{G}_{x_i}(x'')}{\sum_{x''' \in V} \mathcal{G}_{x_i}(x''')} R(X, V \setminus \{x''\}, m-1).$$

In addition, we have:

$$R(X, V, 0) = \begin{cases} 1 & \text{if } X \cap V = \emptyset, \\ 0 & \text{if } X \cap V \neq \emptyset, \end{cases}$$

and $R(X, V, m) = 0$ if $|X \cap V| > m$. When $k_i = 1$ across all users, Eq. (4.2) can be simplified as:

$$\mathbb{P}\big(x' \in H_i \mid x_i\big) = \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i)}. \tag{4.3}$$

When $\mathcal{G}$ assigns an equal, non-zero probability to every candidate in a subset $X_i \subset \mathcal{X} \setminus \{x_i\}$ ($|X_i| \geq k_i$) and probability 0 to all the others (e.g., the chaffing-by-tweaking method [69]), Eq. (4.2) simplifies to:

$$\mathbb{P}\big(x' \in H_i \mid x_i\big) = \begin{cases} \frac{k_i}{|X_i|} & \text{if } x' \in X_i, \\ 0 & \text{if } x' \notin X_i. \end{cases} \tag{4.4}$$

Some existing honeyword-generation algorithms (e.g., [69, 157]) are WSWoR.

4.3.1.2 Bernoulli Sampling

This class of honeyword-generation algorithms independently selects each password candidate as a honeyword with a specified probability. Consequently, the number of honeywords generated per account can be any non-negative integer (i.e., $k_i \in \mathbb{Z}_{\geq 0}$) and may differ from account to account. Formally, for a user i with her user password x_i, given a password generative model $\mathcal{G}$ that assigns a probability $\mathcal{G}_{x_i}(x')$ to a password candidate $x' \in \mathcal{X}$, it independently selects each password candidate $x' \in \mathcal{X} \setminus \{x_i\}$ as honeyword with a probability of $\beta \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i)}$, where $\beta > 0$ is a parameter controlling the expected number of generated

Algorithm 1 Honeyword-generation algorithm

Input: Password generative model $\mathcal{G}$ and user password x_i.

if WSWoR **then**
  $H_i \leftarrow \emptyset$;
  for $= 1, 2, \ldots, k_i$ **do**
    Sample an $x' \in \mathcal{X} \setminus (\{x_i\} \cup H_i)$ according to the weights $\left\{\mathcal{G}_{x_i}(x')\right\}_{x' \in \mathcal{X} \setminus (\{x_i\} \cup H_i)}$ and add it into H_i;
  end for
else if Bernoulli sampling **then**
  $H_i \leftarrow \emptyset$
  for $x' \in \mathcal{X} \setminus \{x_i\}$ **do**
    Add x' into H_i with probability $\beta \frac{\mathcal{G}_{x_i}(x')}{1-\mathcal{G}_{x_i}(x_i)}$;
  end for
end if

Output: H_i

honeywords per account. In other words, for each $x' \in \mathcal{X} \setminus \{x_i\}$, we have:

$$\mathbb{P}\left(x' \in H_i \mid x_i\right) = \beta \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i)}. \tag{4.5}$$

When we set $\beta \frac{\mathcal{G}_{x_i}(x')}{1-\mathcal{G}_{x_i}(x_i)}$ as the same constant for all $x' \in \mathcal{X} \setminus \{x_i\}$ (i.e., we use a $\mathcal{G}$ that assigns equal probability to all candidates), the resulting procedure is exactly the honeyword-generation algorithm used in Bernoulli Honeywords [160]. Thus, our Bernoulli-sampling-based honeyword-generation algorithm can be viewed as a generalization of the Bernoulli-Honeywords approach.

We present the pseudocode of our honeyword-generation algorithm in Alg. 1. Since our honeyword-generation algorithm is compatible with arbitrary password generative models in a WSWoR or a Bernoulli sampling process, our honeyword-generation algorithm subsumes most existing honeyword-generation methods.

4.3.2 Credential-Database Breach Detection

Formulating credential-database breach detection as an online changepoint-detection problem: Our breach-detection algorithm is an online algorithm that detects a breach by sequentially analyzing a stream of login-attempt data, each consisting of a user ID and a password.

We set the time $t = 0$ when the system initializes the breach-detection algorithm. After that, suppose we have a stream of random variables $Y_1, Y_2, \ldots$ that can be observed from the online login attempts. We will introduce the definitions of these random variables later. We assume that under the "normal condition" (i.e., when *no* credential-database breach happens), these variables $Y_1, Y_2, \ldots$ follow a distribution P that we call as "pre-change" distribution. *Here the "normal condition" encompasses all system behaviors when no credential-database breach happens, including the presence of false-alarm attacks.* The distribution of the stream of random variables might change at the time ν that we call *changepoint*. In our problem, ν is the time when the breach attacker starts to try the credentials from the breached database (i.e., sweetwords) to exploit the accounts after the credential-database breach has happened. After *changepoint*, the stream of random variables $Y_{\nu+1}, Y_{\nu+2}, \ldots$ follow a different distribution Q that we call as "post-change" distribution. $\nu = 0$ means that the credential-database breach has happened and the breach attacker has started to try the sweetwords for login attempts when the system initializes the detection mechanism while $\nu = \infty$ means that no credential-database breach ever happens. Our goal is to continuously detect whether a changepoint has occurred given the login-attempt data observed so far.

Random variables measured for detection: We want to measure whether an incorrect login attempt uses a honeyword. An incorrect login attempt is defined as the one in which the submitted password does not match the user password for the corresponding account (e.g., by comparing the salted hash of the submitted password with that of the corresponding user password). Accordingly, each random variable Y indicates whether an incorrect login attempt uses a honeyword and takes values in $\{0, 1\}$. Specifically, given an incorrect login attempt with user ID i and password x', we define:

$$Y = \begin{cases} 1 & \text{if } x' \in H_i^*, \\ 0 & \text{if } x' \notin H_i^*, \end{cases} \tag{4.6}$$

where $H_i^* \subseteq H_i$ represents the current set of honeywords for account i stored in the credential database. As described in Sec. 4.1.1, if a honeyword in H_i has been used in a prior

login attempt, it is removed from the stored set (i.e., excluded from H_i^*) and thus cannot contribute to $Y = 1$ in subsequent attempts until a new honeyword set is generated.

"Pre-change" and "post-change" distributions: Given a stream of random variables $Y_1, Y_2, \ldots$, under "pre-change" distribution P (i.e., $t \leq \nu$), we define:

$$\mathbb{P}_P\big(Y_t = 1 \mid Y_1, Y_2, \ldots, Y_{t-1}\big) = p_t, \tag{4.7}$$

while, under "post-change" distribution Q (i.e., $t > \nu$), we define:

$$\mathbb{P}_Q\big(Y_t = 1 \mid Y_1, Y_2, \ldots, Y_{t-1}\big) = q_t, \tag{4.8}$$

where $p_t, q_t \in [0, 1]$.

In Eq. (4.7), p_t depends on the login attempt, the associated login history C_i, and the honeyword-generation algorithm. Here login history C_i is the set of distinct incorrect passwords that have been attempted for account i since the last (re-)generation of its honeywords. While p_t precisely captures the "pre-change" behavior, computing p_t (Eq. (4.9)) requires maintaining the login history C_i, which is often impractical and may introduce leakage risks. To address this, we will later replace p_t with a computable upper bound (introduced in Eq. (4.12) and Eq. (4.13)) that eliminates the need for login history.

Formally, when the t-th incorrect login attempt tries user ID i and password x', we have the probability p_t in Eq. (4.7) as:

$$p_t = \mathbb{P}\big(x' \in H_i^* \mid x_i, C_i\big), \tag{4.9}$$

where $\mathbb{P}\big(x' \in H_i^* \mid x_i, C_i\big)$ is the probability of x' being selected as a honeyword for the account i conditioned on the information from C_i. If the honeyword-generation algorithm is WSWoR, we have:

$$\mathbb{P}\big(x' \in H_i^* \mid x_i, C_i\big) = \begin{cases} \displaystyle\sum_{\substack{X \subset \mathcal{X} \setminus (\{x_i\} \cup C_i) \\ \text{s.t. } x' \in X, |X| = k_i^*}} R(X, \mathcal{X} \setminus (\{x_i\} \cup C_i), k_i^*) & \text{if } x' \notin C_i, \\ 0 & \text{if } x' \in C_i, \end{cases} \tag{4.10}$$

where $R(\cdot, \cdot, \cdot)$ has been defined in Sec. 4.3.1.1 and $k_i^* \leq k_i$ denotes the number of honeywords remained in H_i^* (i.e., $k_i^* = |H_i^*|$). Eq. (4.10) captures the probability that x' is selected in

one of the remaining k_i^* draws without replacement from the candidate space $\mathcal{X} \setminus (\{x_i\} \cup C_i)$. If $x' \in C_i$, the probability is 0 because it has been excluded from H_i^*.

If the honeyword-generation algorithm is Bernoulli-sampling-based, we have:

$$\mathbb{P}\big(x' \in H_i^* \mid x_i, C_i\big) = \begin{cases} \mathbb{P}\big(x' \in H_i \mid x_i\big) & \text{if } x' \notin C_i, \\ 0 & \text{if } x' \in C_i, \end{cases} \tag{4.11}$$

where $\mathbb{P}\big(x' \in H_i \mid x_i\big)$ has been defined in Eq. (4.5) in Sec. 4.3.1.2. This equality holds because, under Bernoulli sampling, the inclusion of each candidate password is independent. Therefore, conditioning on C_i does not affect the probability that x' is selected as a honeyword, unless $x' \in C_i$, in which case it is excluded from H_i^*.

Under Q (i.e., when a breach attacker tries to use the credentials in the breached database in login attempts to exploit the accounts), we have $q_t = 1$ if considering a "naive" breach attacker who just tries the sweetwords in the breached credential database. However, a strategic breach attacker might apply a strategy to mitigate the detection, e.g., by trying to use the passwords that fall outside the sweetword sets for login attempts. Under such an adaptive breach attack strategy, we would have $q_t < 1$. Consequently, q_t varies with the breach attacker's knowledge and strategy, and is therefore unknown to the site or the breach-detection algorithm. To address this, we will later replace q_t with a computable approximation introduced in Eq. (4.14).

Online credential-database breach detection algorithm: Overall, our breach-detection algorithm is formulated as the design of a detection time $n^* \in \{1, 2, \ldots, \infty\}$. The detection time n^* means that we declare a credential-database breach has happened after observing $Y_1, Y_2, \ldots, Y_{n^*}$. We define $n^* = \infty$ if we *never* detect the breach during the entire execution of the breach-detection algorithm including cases where the algorithm is terminated by the system for external reasons without raising an alarm. When our "pre-change" distribution P and "post-change" distribution Q are known (i.e., p_t in Eq. (4.7) and q_t in Eq. (4.8) are known), we can design our breach-detection algorithm by the classical CUSUM procedure [9, 105, 127], which achieves the optimal detection delay in the sense of minimizing $\max\{n^* - \nu, 0\}$, while providing false-detection guarantees.

In our problem, p_t is well-defined (Eq. (4.9)) but computing the exact p_t by Eq. (4.9) requires knowledge of the corresponding user password and the passwords in the associated login history, which is inapplicable in an authentication system where only salted hashes of passwords are stored and maintaining the login history may increase leakage risks. In addition, q_t varies with the breach attacker's knowledge and strategy, and thus q_t is unknown. Therefore, the classical CUSUM procedure is inapplicable to our problem.

To address this, we upper-bound p_t by a quantity p_t^* such that $p_t^* \geq p_t$ for any t. The bound p_t^* is computed using the password generative model $\mathcal{G}$ employed by the honeyword-generation algorithm, the password x' submitted in the incorrect login attempt, the number k_i^* of honeywords remaining in H_i^*, and the number r_i of incorrect login attempts for account i since the last (re-)generation of its honeywords prior to the t-th incorrect attempt. The computation of p_t^* does not require the knowledge of the user password and passwords in C_i, or maintaining a login history. We denote this computation by $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$.

For honeyword-generation algorithms that use an *input-independent* model, we compute $u^{\mathcal{G}}(x', k_i^*, r_i)$ as:[1]

$$u^{\mathcal{G}}(x', k_i^*, r_i) \leftarrow \begin{cases} \sum_{a=1}^{k_i^*} \frac{\mathcal{G}(x')}{1-(a+r_i)\theta_{\mathcal{G}}} & \text{if WSWoR,} \\ \beta \frac{\mathcal{G}(x')}{1-\theta_{\mathcal{G}}} & \text{if Bernoulli,} \end{cases} \tag{4.12}$$

where $\theta_{\mathcal{G}} = \max_{x'' \in \mathcal{X}} \mathcal{G}(x'')$, i.e., the maximum probability assigned by $\mathcal{G}$ to any candidate password in $\mathcal{X}$. For honeyword-generation algorithms that use an *input-dependent* model, we compute $u^{\mathcal{G}}(x', k_i^*, r_i)$ as:

$$u^{\mathcal{G}}(x', k_i^*, r_i) \leftarrow \begin{cases} \sum_{a=1}^{k_i^*} \frac{\theta_{\mathcal{G}}}{1-(a+r_i)\theta_{\mathcal{G}}} & \text{if WSWoR,} \\ \beta \frac{\theta_{\mathcal{G}}}{1-\theta_{\mathcal{G}}} & \text{if Bernoulli,} \end{cases} \tag{4.13}$$

where $\theta_{\mathcal{G}} = \max_{x'', x''' \in \mathcal{X}} \mathcal{G}_{x''}(x''')$, i.e., the maximum probability assigned by $\mathcal{G}$ over all conditioning passwords and candidate passwords in $\mathcal{X}$. For the WSWoR case, positivity of $u^{\mathcal{G}}(x', k_i^*, r_i)$ requires $1 - (k_i^* + r_i)\theta_{\mathcal{G}} > 0$. This can be enforced by capping the maximum probability assigned by $\mathcal{G}$ at a sufficiently small value (see Sec. 4.4.2.2) and requiring the

[1] Because $\mathcal{G}$ is input-independent in this case, we omit its conditioning argument when writing $\mathcal{G}(x')$ to avoid confusion that the computation of $u^{\mathcal{G}}(x', k_i^*, r_i)$ depends on x_i.

user to reset her account when r_i grows too large. We prove in Sec. 4.3.3 that computing p_t^* using Eq. (4.12) or Eq. (4.13) enforces $p_t^* \geq p_t$.

In addition, we estimate q_t based on the previous measurements of $Y_1, Y_2, \ldots, Y_{t-1}$, before we observe Y_t. Let q_t^* denote the estimation of q_t. We apply a simple estimator and restrict the estimation to the most recent W login-attempt data (i.e., $Y_{\max\{t-W,1\}}, Y_{t-W+1}, \ldots, Y_{t-1}$). This is because those earlier login-attempt data might follow the "pre-change" distribution, whereas more recent login attempts, if under the "post-change" distribution, would provide more information on the breach attacker's evolving strategy. As such, we estimate q_t (for $t > 1$) with Laplace smoothing as

$$q_t^* \leftarrow \frac{\sum_{t'=\max\{t-W,1\}}^{t-1} Y_{t'} + 1}{\min\{t-1, W\} + 2}. \tag{4.14}$$

Based on these constructions, we design our breach-detection algorithm as:

1) Initialization: We set $M_0 \leftarrow 0$, $n^* \leftarrow \infty$, and $q_1^* \leftarrow \frac{1}{2}$. For all $i = 1, \ldots, n$, we set $k_i^* \leftarrow k_i$ and $r_i \leftarrow 0$.
2) Update: When the t-th incorrect login attempt (with user ID i and password x') arrives, we measure Y_t by Eq. (4.6). We compute $p_t^* \leftarrow \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$ and $M_t \leftarrow \max\{M_{t-1}, \omega_t\} \cdot I_t(Y_t)$ where

$$I_t(Y_t) \leftarrow \begin{cases} \frac{1-\max\{q_t^*, p_t^*\}}{1-p_t^*} & \text{if } Y_t = 0, \\ \frac{\max\{q_t^*, p_t^*\}}{p_t^*} & \text{if } Y_t = 1, \end{cases} \tag{4.15}$$

and $\omega_t \leftarrow \frac{\ln 2}{(t+2)(\ln(t+2))^2}$. Finally, we increment $r_i \leftarrow r_i + 1$, and if $Y_t = 1$, we decrement $k_i^* \leftarrow k_i^* - 1$.
3) Termination: If $M_t \geq \frac{1}{\alpha}$, we stop the detection algorithm and declare that the credential-database breach has happened, i.e., we set n^* as the time step t when we stop. Otherwise, we continue, set q_{t+1}^* by Eq. (4.14), and wait for the next incorrect login attempt.

We present the pseudocode of our credential-database-breach detection algorithm in Alg. 2. Although our detection algorithm does not achieve the optimality of detection delay due to

Algorithm 2 Breach-detection algorithm

Input: Password generative model $\mathcal{G}$, window size W, and the false-detection bound α.

$n^* \leftarrow \infty$;
$M_0 \leftarrow 0$;
$t \leftarrow 0$;
$q_1^* \leftarrow \frac{1}{2}$;
for $i = 1, 2, \ldots, n$ **do**
  $k_i^* \leftarrow k_i$;
  $r_i \leftarrow 0$;
end for
while $n^* = \infty$ **do**
  if an incorrect login attempt (with user ID i and password x') arrives **then**
    $t \leftarrow t + 1$;
    $p_t^* \leftarrow \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$;
    $r_i \leftarrow r_i + 1$;
    if $x' \in H_i^*$ **then**
      $Y_t \leftarrow 1$;
      $k_i^* \leftarrow k_i^* - 1$;
    else
      $Y_t \leftarrow 0$;
    end if
    $M_t \leftarrow \max\{M_{t-1}, \omega_t\} \cdot I_t(Y_t)$, where $I_t(Y_t)$ is defined by Eq. (4.15) and $\omega_t \leftarrow \frac{\ln 2}{(t+2)(\ln(t+2))^2}$;
    if $M_t \geq \frac{1}{\alpha}$ **then**
      $n^* \leftarrow t$;
    else
      Estimate q_{t+1}^* on $Y_{\max\{t-W+1,1\}}, \ldots, Y_t$;
    end if
  end if
end while

Output: n^*

its need to use an upper bound on p_t and to estimate q_t, it still achieves a provably bounded global false-detection rate. We will prove false-detection guarantee in the next section (i.e., Sec. 4.3.3).

4.3.3 Guarantee on False Detection

We first prove that computing $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$ where $u^{\mathcal{G}}(x', k_i^*, r_i)$ is computed using Eq. (4.12) or Eq. (4.13) enforces $p_t^* \geq p_t$. After establishing this bound, we present several necessary propositions that form the foundation for our main proof regarding the

false-detection rate.

Proposition 1 (Upper-Bounding p_t)**.** *Setting* $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$ *where* $u^{\mathcal{G}}(x', k_i^*, r_i)$ *is computed by Eq. (4.12) when* $\mathcal{G}$ *is input-independent or Eq. (4.13) when* $\mathcal{G}$ *is input-dependent, for all* t*, we enforce that:*

$$p_t \leq p_t^*. \tag{4.16}$$

Proof. Consider the t-th incorrect login attempt, which uses user ID i and password x'. In this proof, it suffices to consider the case where $x' \notin C_i$, because when $x' \in C_i$, we have $p_t = 0$, and thus any p_t^* (including $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$) trivially upper-bounds p_t. When the honeyword-generation algorithm is the Bernoulli-sampling one, by Eq. (4.5), Eq. (4.9), Eq. (4.11), and $\theta_{\mathcal{G}} = \max_{x'', x''' \in \mathcal{X}} \mathcal{G}_{x''}(x''')$, we have:

$$p_t = \beta \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i)} \leq \beta \frac{\mathcal{G}_{x_i}(x')}{1 - \theta_{\mathcal{G}}} \leq \beta \frac{\theta_{\mathcal{G}}}{1 - \theta_{\mathcal{G}}}. \tag{4.17}$$

Because p_t is a probability, we also have $p_t \leq 1$. Therefore, when we set $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$ and $u^{\mathcal{G}}(x', k_i^*, r_i)$ is computed by Eq. (4.12) when $\mathcal{G}$ is input-independent or Eq. (4.13) when $\mathcal{G}$ is input-dependent, we enforce that $p_t \leq p_t^*$.

When the honeyword-generation algorithm is the WSWoR one that generates H_i through k_i successive draws without replacement, according to Eq. (4.9) and Eq. (4.10), p_t is the probability that x' is selected in one of the remaining k_i^* draws without replacement conditioned that the user password is x_i and the candidate selection space is $\mathcal{X} \setminus (\{x_i\} \cup C_i)$. We use Ω_{a-1} to denote the set of passwords that are selected at the first $a-1$ remaining draws. We have:

$$p_t = \sum_{a=1}^{k_i^*} \mathbb{P}\big(x' \notin \Omega_{a-1} \mid x_i, C_i\big) \tag{4.18}$$

$$\times\, \mathbb{P}\big(x' \text{ is selected at } a\text{-th draw} \mid x' \notin \Omega_{a-1}, x_i, C_i\big) \tag{4.19}$$

$$\leq \sum_{a=1}^{k_i^*} \mathbb{P}\big(x' \text{ is selected at } a\text{-th draw} \mid x' \notin \Omega_{a-1}, x_i, C_i\big) \tag{4.20}$$

Here we have:

$$\mathbb{P}\big(x' \text{ is selected at } a\text{-th draw} \mid x' \notin \Omega_{a-1}, x_i, C_i\big) \tag{4.21}$$

$$= \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i) - \sum_{x'' \in \Omega_{a-1}} \mathcal{G}_{x_i}(x'') - \sum_{x'' \in C_i} \mathcal{G}_{x_i}(x'')}. \tag{4.22}$$

Because $\mathcal{G}_{x_i}(x_i) \leq \theta_{\mathcal{G}}$, $\sum_{x'' \in \Omega_{a-1}} \mathcal{G}_{x_i}(x'') \leq (a-1)\theta_{\mathcal{G}}$, $\sum_{x'' \in C_i} \mathcal{G}_{x_i}(x'') \leq r_i \theta_{\mathcal{G}}$, and we can enforce $1 - (a + r_i)\theta_{\mathcal{G}} > 0$ as described in Sec. 4.3.2, we have:

$$\mathbb{P}\big(x' \text{ is selected at } a\text{-th draw} \mid x' \notin \Omega_{a-1}, x_i, C_i\big) \tag{4.23}$$

$$= \frac{\mathcal{G}_{x_i}(x')}{1 - \mathcal{G}_{x_i}(x_i) - \sum_{x'' \in \Omega_{a-1}} \mathcal{G}_{x_i}(x'') - \sum_{x'' \in C_i} \mathcal{G}_{x_i}(x'')} \tag{4.24}$$

$$\leq \frac{\mathcal{G}_{x_i}(x')}{1 - (a + r_i)\theta_{\mathcal{G}}}. \tag{4.25}$$

As such, we have:

$$p_t \leq \sum_{a=1}^{k_i^*} \mathbb{P}\big(x' \text{ is selected at } a\text{-th draw} \mid x' \notin \Omega_{a-1}, x_i, C_i\big) \tag{4.26}$$

$$\leq \sum_{a=1}^{k_i^*} \frac{\mathcal{G}_{x_i}(x')}{1 - (a + r_i)\theta_{\mathcal{G}}} \leq \sum_{a=1}^{k_i^*} \frac{\theta_{\mathcal{G}}}{1 - (a + r_i)\theta_{\mathcal{G}}}. \tag{4.27}$$

Because p_t is a probability, we also have $p_t \leq 1$. Therefore, when we set $p_t^* = \min\{u^{\mathcal{G}}(x', k_i^*, r_i), 1\}$ and $u^{\mathcal{G}}(x', k_i^*, r_i)$ is computed by Eq. (4.12) when $\mathcal{G}$ is input-independent or Eq. (4.13) when $\mathcal{G}$ is input-dependent, we enforce that $p_t \leq p_t^*$ in this case. □

Proposition 2. *Let $I_t(Y_t)$ computed by Eq. (4.15). For any t, by enforcing $p_t \leq p_t^*$, we have:*

$$\mathbb{E}_P\left[I_t(Y_t) \mid Y_1, \ldots, Y_{t-1}\right] \leq 1. \tag{4.28}$$

Proof. According to Eq. (4.15) and Eq. (4.7), we have:

$$\mathbb{E}_P\left[I_t(Y_t) \mid Y_1, \ldots, Y_{t-1}\right] \tag{4.29}$$

$$= I_t(Y_t = 1) \cdot \mathbb{P}_P\big(Y_t = 1 \;\big|\; Y_1, Y_2, \ldots, Y_{t-1}\big) \tag{4.30}$$

$$+ I_t(Y_t = 0) \cdot \mathbb{P}_P\big(Y_t = 0 \;\big|\; Y_1, Y_2, \ldots, Y_{t-1}\big) \tag{4.31}$$

$$= \frac{\max\{q_t^*, p_t^*\}}{p_t^*} p_t + \frac{1 - \max\{q_t^*, p_t^*\}}{1 - p_t^*}(1 - p_t) \tag{4.32}$$

$$= \left(\frac{\max\{q_t^*, p_t^*\}}{p_t^*} - \frac{1 - \max\{q_t^*, p_t^*\}}{1 - p_t^*}\right) p_t + \frac{1 - \max\{q_t^*, p_t^*\}}{1 - p_t^*} \tag{4.33}$$

$$= \frac{\max\{q_t^*, p_t^*\} - p_t^*}{(1 - p_t^*)p_t^*} p_t + \frac{1 - \max\{q_t^*, p_t^*\}}{1 - p_t^*}, \tag{4.34}$$

which is linear in p_t. By enforcing $p_t \leq p_t^*$, we have $\frac{\max\{q_t^*, p_t^*\} - p_t^*}{(1 - p_t^*)p_t^*} \geq 0$, and thus it is maximized at 1 that occurs when $p_t = p_t^*$. In other words, we have:

$$\frac{\max\{q_t^*, p_t^*\} - p_t^*}{(1 - p_t^*)p_t^*} p_t + \frac{1 - \max\{q_t^*, p_t^*\}}{1 - p_t^*} \leq 1, \tag{4.35}$$

which makes our proof. □

Proposition 3. *Let G be a discrete random variable and let $Y_1, Y_2, \ldots, Y_{t-1}$ be random variables taking values in $\{0, 1\}$. We have:*

$$\mathbb{E}_P\left[G\right] = \mathbb{E}_P\left[\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t-1}\right]\right]. \tag{4.36}$$

Proof.

$$\mathbb{E}_P\left[\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t-1}\right]\right] \tag{4.37}$$

$$= \sum_{y_1} \cdots \sum_{y_{t-1}} \mathbb{E}_P\left[G \mid Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right]$$

$$\times \mathbb{P}_P\left(Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right) \tag{4.38}$$

$$= \sum_{y_1} \cdots \sum_{y_{t-1}} \sum_{g} g\mathbb{P}_P\big(G = g \;\big|\; Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\big)$$

$$\times \mathbb{P}_P\left(Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right) \tag{4.39}$$

$$= \sum_{g} g\mathbb{P}_P\left(G = g\right) \tag{4.40}$$

$$= \mathbb{E}_P\left[G\right]. \tag{4.41}$$

□

Proposition 4. *Let G be a discrete random variable and let $Y_1, Y_2, \ldots, Y_{t-1}$ be random variables taking values in $\{0, 1\}$. For any $t' \leq t$, we have:*

$$\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t'-1}\right] = \mathbb{E}_P\left[\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t-1}\right] \mid Y_1, \ldots, Y_{t'-1}\right]. \tag{4.42}$$

Proof. Given that $Y_1 = y_1, \ldots, Y_{t'-1} = y_{t'-1}$, we have:

$$\mathbb{E}_P\left[\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t-1}\right] \mid Y_1 = y_1, \ldots, Y_{t'-1} = y_{t'-1}\right] \tag{4.43}$$

$$= \sum_{y_{t'}} \cdots \sum_{y_{t-1}} \mathbb{E}_P\left[G \mid Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right]$$

$$\times \mathbb{P}_P\left(\begin{matrix} Y_{t'} = y_{t'}, \ldots, \\ Y_{t-1} = y_{t-1} \end{matrix} \;\middle|\; \begin{matrix} Y_1 = y_1, \ldots, \\ Y_{t'-1} = y_{t'-1} \end{matrix}\right) \tag{4.44}$$

$$= \sum_{y_{t'}} \cdots \sum_{y_{t-1}} \sum_{g} g\mathbb{P}_P\big(G = g \;\big|\; Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\big)$$

$$\times \mathbb{P}_P\left(\begin{matrix} Y_{t'} = y_{t'}, \ldots, \\ Y_{t-1} = y_{t-1} \end{matrix} \;\middle|\; \begin{matrix} Y_1 = y_1, \ldots, \\ Y_{t'-1} = y_{t'-1} \end{matrix}\right) \tag{4.45}$$

$$= \sum_{g} g\mathbb{P}_P\big(G = g \;\big|\; Y_1 = y_1, \ldots, Y_{t'-1} = y_{t'-1}\big) \tag{4.46}$$

$$= \mathbb{E}_P\left[G \mid Y_1 = y_1, \ldots, Y_{t'-1} = y_{t'-1}\right]. \tag{4.47}$$

□

Proposition 5. *Let G be a discrete random variable, let $Y_1, Y_2, \ldots, Y_{t-1}$ be random variables taking values in $\{0,1\}$, and let $f(Y_1, \ldots, Y_{t-1})$ be a function of $Y_1, \ldots, Y_{t-1}$. We have:*

$$\mathbb{E}_P\left[G \cdot f(Y_1, \ldots, Y_{t-1}) \mid Y_1, \ldots, Y_{t-1}\right] = f(Y_1, \ldots, Y_{t-1})\mathbb{E}_P\left[G \mid Y_1, \ldots, Y_{t-1}\right] \tag{4.48}$$

Proof. Given that $Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}$, we have:

$$\mathbb{E}_P\left[G \cdot f(y_1, \ldots, y_{t-1}) \mid Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right] \tag{4.49}$$

$$= \sum_g g f(y_1, \ldots, y_{t-1}) \mathbb{P}_P\big(G = g \;\big|\; Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\big) \tag{4.50}$$

$$= f(y_1, \ldots, y_{t-1}) \sum_g g \mathbb{P}_P\big(G = g \;\big|\; Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\big) \tag{4.51}$$

$$= f(y_1, \ldots, y_{t-1})\mathbb{E}_P\left[G \mid Y_1 = y_1, \ldots, Y_{t-1} = y_{t-1}\right]. \tag{4.52}$$

□

We use *false-detection rate* to quantify false alarms of our breach-detection algorithm. False-detection rate is the probability that the detection time of the breach-detection algorithm is finite (i.e., it declares a credential-database breach) under the "pre-change" distribution. Equivalently, it is the probability that the algorithm ever raises a breach detection over the infinite login attempts during its entire execution, when no credential-database breach happens. We have the following theorem on the false-detection rate of our method:

Theorem 6 (False-detection rate)**.** *By enforcing $q_t^* \geq p_t^*$ for all t, the false-detection rate of our breach-detection algorithm is upper bounded by α. In other words, when no credential-database breach happens (i.e., under the "pre-change" distribution P), we have:*

$$\mathbb{P}_P\left(n^* < \infty\right) < \alpha. \tag{4.53}$$

Proof. First, since $M_t = \max\{M_{t-1}, \omega_t\} \cdot I_t(Y_t)$ and $M_0 = 0$, we have:

$$M_{n^*} = \max_{t \in \{1,\ldots,n^*\}} \omega_t \prod_{t'=t}^{n^*} I_{t'}(Y_{t'}). \tag{4.54}$$

Moreover, we have:

$$\max_{t \in \{1,\ldots,n^*\}} \omega_t \prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \leq \sum_{t=1}^{n^*} \omega_t \prod_{t'=t}^{n^*} I_{t'}(Y_{t'}), \tag{4.55}$$

which implies:

$$\mathbb{E}_P\left[M_{n^*}\right] \leq \mathbb{E}_P\left[\sum_{t=1}^{n^*} \omega_t \prod_{t'=t}^{n^*} I_{t'}(Y_{t'})\right] = \sum_{t=1}^{n^*} \omega_t \mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'})\right]. \tag{4.56}$$

By Prop. 3 where the discrete random variable G is $\prod_{t'=t}^{n^*} I_{t'}(Y_{t'})$, we have:

$$\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'})\right] = \mathbb{E}_P\left[\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right]\right], \tag{4.57}$$

where $Y_{1:t-1}$ represents $Y_1, \ldots, Y_{t-1}$. Using Prop. 4 where the discrete random variable G is $\prod_{t'=t}^{n^*} I_{t'}(Y_{t'})$, we have:

$$\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right] = \mathbb{E}_P\left[\mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \cdot I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \mid Y_{1:t-1}\right]. \tag{4.58}$$

By Prop. 5 with the discrete random variable G being $I_{n^*}(Y_{n^*})$ and $\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'})$ is a function of $Y_{1:n^*-1}$, we have:

$$\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right] \tag{4.59}$$

$$= \mathbb{E}_P\left[\mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \cdot I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \mid Y_{1:t-1}\right] \tag{4.60}$$

$$= \mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \cdot \mathbb{E}_P\left[I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \mid Y_{1:t-1}\right]. \tag{4.61}$$

By Prop. 2,

$$\mathbb{E}_P\left[I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \leq 1. \tag{4.62}$$

Therefore,

$$\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right] \tag{4.63}$$

$$= \mathbb{E}_P\left[\mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \cdot I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \mid Y_{1:t-1}\right] \tag{4.64}$$

$$= \mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \cdot \mathbb{E}_P\left[I_{n^*}(Y_{n^*}) \mid Y_{1:n^*-1}\right] \mid Y_{1:t-1}\right] \tag{4.65}$$

$$\leq \mathbb{E}_P\left[\prod_{t'=t}^{n^*-1} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right] \tag{4.66}$$

Iterating this argument backward down to t and by Prop. 2, we obtain:

$$\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right] \leq \mathbb{E}_P\left[I_t(Y_t) \mid Y_{1:t-1}\right] \leq 1. \tag{4.67}$$

Therefore,

$$\mathbb{E}_P\left[M_{n^*}\right] \leq \sum_{t=1}^{n^*} \omega_t \mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'})\right] \tag{4.68}$$

$$= \sum_{t=1}^{n^*} \omega_t \mathbb{E}_P\left[\mathbb{E}_P\left[\prod_{t'=t}^{n^*} I_{t'}(Y_{t'}) \mid Y_{1:t-1}\right]\right] \tag{4.69}$$

$$\leq \sum_{t=1}^{n^*} \omega_t. \tag{4.70}$$

Since $\omega_t = \frac{\ln 2}{(t+2)(\ln(t+2))^2}$, we have:

$$\mathbb{E}_P\left[M_{n^*}\right] \leq \sum_{t=1}^{n^*} \frac{\ln 2}{(t+2)(\ln(t+2))^2} \tag{4.71}$$

$$= \sum_{t=3}^{n^*+2} \frac{\ln 2}{t(\ln t)^2} \tag{4.72}$$

$$< \int_2^{\infty} \frac{\ln 2}{t(\ln t)^2} dt \tag{4.73}$$

$$= 1. \tag{4.74}$$

Finally, because M_t is non-negative for all t, we prove:

$$\mathbb{P}_P\left(n^* < \infty\right) = \mathbb{P}_P\left(M_{n^*} \geq \frac{1}{\alpha}\right) \tag{4.75}$$

$$\leq \alpha \cdot \mathbb{E}_P\left[M_{n^*}\right] \tag{4.76}$$

$$< \alpha, \tag{4.77}$$

where the equality comes from the definition of n^* that $n^* = \inf\{t \in \{1, 2, \dots\} : M_t \geq \frac{1}{\alpha}\}$ and the first inequality comes from Markov's inequality [99, p. 44]. □

We emphasize that the false-detection rate we consider is *a global false-detection rate*. In addition, as we have mentioned in Sec. 4.3.2, the "pre-change" distribution P captures all system behaviors when no credential-database breach occurs, including potential false-alarm attacks. Therefore, this bound ensures that, regardless of the number or assumptions of false-alarm attacks, the probability that the breach-detection algorithm ever raises a false detection remains provably bounded.

4.4 Experiments

In this section, we empirically evaluate the robustness of our method against false-alarm attacks and its performance on detecting credential-database breaches under a wide range of attacks.

4.4.1 Attack Strategies

4.4.1.1 False-Alarm Attack

As mentioned in Sec. 4.2.1, we assume that the false-alarm attacker knows 1) a subset of user passwords $\{x_i\}_{i \in A}$ (e.g., by registering accounts at the site) and 2) the honeyword-generation algorithm (i.e., the password generative model $\mathcal{G}$) and the breach-detection algorithm used by the site. To falsely launch the site's alarm, the attacker tries an ordered sequence of login attempts denoted as $F = \left(F_1, F_2, \dots\right)$, where each attempt F_b is a pair of user ID $F_b.\mathsf{ID} \in \{1, 2, \dots, n\}$ and password $F_b.\mathsf{PW} \in \mathcal{X}$. The attack is subject to the following constraint: the maximum number of false-alarm attacker's login attempts per account is bounded by γ, i.e., $|\{F_b \mid b = 1, 2, \cdots \wedge F_b.\mathsf{ID} = i\}| \leq \gamma, \forall i = 1, 2, \dots, n$. Considering a

breadth-first attacker [46, Table 5], we set $\gamma = 10^4$ and $|A| = 1{,}000$. We heuristically design the false-alarm attack by following the previous works (e.g., [61]): For each $i \in A$, the attacker selects a set $O_i \subset \{(i, x') \mid x' \in \mathcal{X} \setminus \{x_i\}\}$ by solving

$$O_i \leftarrow \underset{O_i : |O_i| = \gamma}{\arg\max} \sum_{(i,x') \in O_i} \mathcal{G}_{x_i}(x').$$

Then he takes the union by $O \leftarrow \bigcup_{i \in A} O_i$, sorts all attempts in O in a decreasing order of the password probability assigned by $\mathcal{G}$ as an ordered sequence F, and tries the login attempts in F sequentially.

4.4.1.2 Breach Attack

As introduced in Sec. 4.2.1, we assume that the breach attacker knows 1) the distribution of user passwords at the site or the passwords from the same users at other sites, and 2) both the honeyword-generation algorithm and the breach-detection algorithm used by the site. We denote the knowledge of the distribution of user passwords at the site as $\mathcal{B}_1$ and the knowledge of the passwords from the same users at other sites as $\mathcal{B}_2$. Under $\mathcal{B}_2$, we further denote the set of passwords from the user i at other sites as U_i. $\mathcal{B}_1$ is commonly assumed in prior research on honeywords (e.g., [156, 157]) while $\mathcal{B}_2$ represents a substantially stronger adversarial capability and poses the greatest threat to the honeyword security as indicated in Chapter 3. Some previous works (e.g., [157]) also considered breach attackers who know the users' PII and uses this information to distinguish user passwords from honeywords. This type of adversarial knowledge could be subsumed by $\mathcal{B}_2$ because the passwords from the same user at other sites might embed the user's PII.

We denote the breach attacker's login-attempt sequence as $B = (B_1, B_2, \dots)$, where each login attempt is a pair of user ID and password. B is subject to the following constraint: for each $i = 1, 2, \dots, n$, at most γ login attempts in B are on account i. We set $\gamma = 10^4$. We consider two types of breach attackers: 1) a naive attacker who restricts login attempts to sweetwords only, i.e., $B_b.\mathsf{PW} \in S_{B_b.\mathsf{ID}}$ for all b, and 2) an adaptive attacker who tries both sweetwords and passwords outside the corresponding sweetword sets with the goal of

evading or mitigating the site's breach detection.

Naive breach attacker: We design two heuristic attack strategies for a naive breach attacker: 1) likelihood-based greedy strategy and 2) reward-cost-ratio-based greedy strategy. The likelihood-based greedy strategy prioritizes the sweetwords with higher estimated likelihoods of being user passwords. It was originally proposed by Wang et al. [156] under $\mathcal{B}_1$ and we adapt it here to operate under $\mathcal{B}_2$. It proceeds as follows:

1) Initialization: The breach attacker initializes the sequence $B \leftarrow ()$ and assigns a likelihood $l_{i,j}$ to each $s_{i,j} \in \bigcup_{i'} S_{i'}$. Here $l_{i,j}$ refers to $s_{i,j}$'s likelihood of being the user password. He estimates

$$l_{i,j} \leftarrow \begin{cases} \frac{\mathbb{P}_{\mathcal{B}_1}\left(s_{i,j}\right) \prod_{j'' \neq j} \mathbb{P}\left(s_{i,j''} \in H_i \;\middle|\; s_{i,j}\right)}{\sum_{j'} \mathbb{P}_{\mathcal{B}_1}\left(s_{i,j'}\right) \prod_{j'' \neq j'} \mathbb{P}\left(s_{i,j''} \in H_i \;\middle|\; s_{i,j'}\right)} & \text{if } \mathcal{B}_1, \\ \frac{\max\{\mathsf{sim}(s_{i,j}, x')\}_{x' \in U_i}}{\sum_{s_{i,j'} \in S_i} \max\{\mathsf{sim}(s_{i,j'}, x')\}_{x' \in U_i}} & \text{if } \mathcal{B}_2, \end{cases}$$

where $\mathbb{P}_{\mathcal{B}_1}\left(s_{i,j}\right)$ is the probability that $s_{i,j}$ is the user password under the knowledge of $\mathcal{B}_1$, $\mathbb{P}\left(s_{i,j''} \in H_i \;\middle|\; s_{i,j}\right)$ is the probability that $s_{i,j''}$ is a honeyword conditioned that $s_{i,j}$ is the user password, and sim is a password similarity metric (which we will introduce in Sec. 4.4.2.3).

2) Greedy selection and update: At each step b, he selects an $(i, s_{i,j})$ by solving $\arg\max_{(i,j)} l_{i,j}$ and tries $(i, s_{i,j})$ as B_b for login attempt. He updates $B \leftarrow B \,\|\, (B_b)$ and $l_{i,j} \leftarrow 0$. If $(i, s_{i,j})$ succeeds (i.e., $s_{i,j} = x_i$), he updates $l_{i,j'} \leftarrow 0$ for all $s_{i,j'} \in S_i \setminus (S_i \cap \{B_{b'}.\mathsf{PW} \mid b' = 1, \ldots, b \wedge B_{b'}.\mathsf{ID} = i\})$ since no further guesses for this account are needed; Otherwise, he renormalizes the remaining likelihoods for user i as $l_{i,j'} \leftarrow \frac{l_{i,j'}}{1 - l_{i,j}}$.

The reward-cost-ratio-based greedy strategy accounts for both the expected benefit and the expected detection risk of a login attempt, and prioritizes sweetwords with higher reward-cost ratios. Here the reward of a sweetword refers to its likelihood of yielding access permission (i.e., the probability that it is the user password), while the cost of a sweetword refers to the expected increment to the detection statistic M_t induced by a login attempt using that sweetword. Specifically, a login attempt using sweetword $s_{i,j}$ either succeeds (if it is the

user password) and causes no change to M_t, or fails (if it is a honeyword) and increments M_t by $I_t(Y_t)$ with $Y_t = 1$. When $s_{i,j}$ is a honeyword, the induced increment $I_t(Y_t)$ with $Y_t = 1$ is proportional to $\frac{1}{\mathcal{G}(s_{i,j})}$ if $\mathcal{G}$ is input-independent or $\frac{1}{\theta_{\mathcal{G}}}$ if $\mathcal{G}$ is input-dependent, according to Eq. (4.15) and $p_t^* = \min\{u^{\mathcal{G}}(s_{i,j}, k_i^*, r_i), 1\}$ where $u^{\mathcal{G}}(s_{i,j}, k_i^*, r_i)$ is computed using Eq. (4.12) or Eq. (4.13). Taking expectation over the attacker's belief about which sweetword is the user password, we define the cost of $s_{i,j}$ as $-(1 - l_{i,j}) \log \mathcal{G}(s_{i,j})$ if $\mathcal{G}$ is input-independent or $-(1 - l_{i,j}) \log \theta_{\mathcal{G}}$ if $\mathcal{G}$ is input-dependent where $l_{i,j}$ is estimated probability that $s_{i,j}$ is the user password. As such, this greedy strategy proceeds as follows:

1) Initialization: The breach attacker initializes $B \leftarrow ()$ and assigns a likelihood $l_{i,j}$ and a cost $c_{i,j}$ to each $s_{i,j} \in \bigcup_i S_i$. Here

$$l_{i,j} \leftarrow \begin{cases} \frac{\mathbb{P}_{\mathcal{B}_1}(s_{i,j}) \prod_{j'' \neq j} \mathbb{P}\left(s_{i,j''} H_i \;\middle|\; s_{i,j}\right)}{\sum_{j'} \mathbb{P}_{\mathcal{B}_1}(s_{i,j'}) \prod_{j'' \neq j'} \mathbb{P}\left(s_{i,j''} H_i \;\middle|\; s_{i,j'}\right)} & \text{if } \mathcal{B}_1, \\ \frac{\max\{\mathsf{sim}(s_{i,j}, x')\}_{x' \in U_i}}{\sum_{s_{i,j'} \in S_i} \max\{\mathsf{sim}(s_{i,j'}, x')\}_{x' \in U_i}} & \text{if } \mathcal{B}_2, \end{cases}$$

$$c_{i,j} \leftarrow \begin{cases} -(1 - l_{i,j}) \log \mathcal{G}(s_{i,j}) & \text{if input-indep. } \mathcal{G}, \\ -(1 - l_{i,j}) \log \theta_{\mathcal{G}} & \text{if input-dep. } \mathcal{G}. \end{cases}$$

2) Greedy selection and update: At each step b, he selects an $(i, s_{i,j})$ by $\arg\max_{(i,j)} \frac{l_{i,j}}{c_{i,j}}$ and tries $(i, s_{i,j})$ as B_b for login attempt. He updates $B \leftarrow B \parallel (B_b)$ and $l_{i,j} \leftarrow 0$. If $(i, s_{i,j})$ succeeds (i.e., $s_{i,j} = x_i$), he updates $l_{i,j'} \leftarrow 0$ for all $s_{i,j'} \in S_i \setminus (S_i \cap \{B_{b'}.\mathsf{PW} \mid b' = 1, \ldots, b \wedge B_{b'}.\mathsf{ID} = i\})$; Otherwise, he renormalizes $l_{i,j'} \leftarrow \frac{l_{i,j'}}{1 - l_{i,j}}$. Then he updates the corresponding cost $c_{i,j'}$.

Adaptive breach attacker: The adaptive breach attacker additionally issues login attempts using the passwords that fall outside the sweetword sets to mitigate the breach detection. Such login attempts may decrease the detection statistic M_t, and can also reduce the estimated parameter q_t^*, thereby lowering the increment induced by subsequent honeyword-based login attempts. We design two heuristic adaptive breach-attack strategies by modifying the likelihood-based greedy method and the reward-cost-ratio-based greedy method described above. Specifically, before attempting a sweetword $s_{i,j}$, the adaptive breach attacker inserts

$\lfloor \frac{\gamma - k_i^* - 1}{k_i^*} \rfloor$ login attempts using passwords drawn from $\mathcal{X} \setminus S_i$. Note that these auxiliary attempts are intended solely to suppress the detection statistic but do not aim to obtain account access.

4.4.2 Experimental Setup

4.4.2.1 The Dataset

The dataset used in our experiments is the *4iQ* password dataset [18], which has been introduced in Sec. 3.2.3.1. The 4iQ dataset has been widely used in prior research on password [61, 106, 170], particularly on how password reuse behaviors across sites impact security [61, 106, 170].

We preprocessed the 4iQ dataset, as introduced in Sec. 3.2.3.1. After the preprocessing, the resulted dataset had 195,894,983 password multisets. We split the dataset into three disjoint subsets comprising 40% 40%, 20% of the data, denoted as D_1, D_2, and D_3, respectively. D_1 was used to train password generative models that the site can leverage to generate honeywords. D_2 was used by the breach attacker to train machine-learning models that can be exploited to distinguish user passwords from honeywords in a breached credential database. D_3 was used to evaluate the performance of our approach. For each multiset in D_3, we randomly selected one password as the user password at the site and treated the remaining passwords as those used by the same user at other sites. Consequently, in each experiment, we constructed a set of user passwords at the site of size n, and a corresponding set of password multisets from the same users at other sites that may be known to the breach attacker. This setup simulated a scenario in which approximately 43% of users reused their passwords across sites. We set $n = 10^7$ by default.

4.4.2.2 Honeyword Generation

We considered a variety of password generative models as $\mathcal{G}$ for honeyword generation. As discussed in Sec. 4.1.2, these models can be broadly categorized into input-independent and input-dependent models. Each category can be further divided into machine-learning-based models and heuristically constructed models. We trained these machine-learning

models on D_1. In our experiments, we evaluated the following password generative models. Our criteria for selecting these password generative models were threefold: (i) the models are representative and widely used in prior research on honeyword generation [61, 69, 157], (ii) they are compatible with WSWoR or Bernoulli sampling, and (iii) their code is publicly available.

- Input-independent, machine-learning models: These models learn a password distribution from a (publicly available) password dataset, e.g., list model [157], PCFG [167], Markov model [41, 90], RNN [96], and their combinations [157]. Since these models have been shown to achieve comparable performance in honeyword generation [61, 157], in our experiments we focused on three representative choices: the PCFG model, the Markov model, and the RNN model. We denote these three models as pcfg, mkv, and rnn respectively.
- Input-independent, heuristically constructed models: We considered a random generator, denoted rgm, that assigns an equal probability to every password in the set $\mathcal{X}_{D_1} \subset \mathcal{X}$. Here $\mathcal{X}_{D_1}$ represents the set of distinct passwords appearing in D_1. The random generator has been used in prior work on honeyword generation, e.g., Bernoulli Honeywords [160]. However, unlike our setting, Bernoulli Honeywords samples from the entire password space $\mathcal{X}$, rather than restricting to the subset $\mathcal{X}_{D_1}$.
- Input-dependent, machine-learning models: These models learn patterns of partial password reuse across different sites from a (publicly available) password dataset. They leverage deep neural networks to transform a password into a similar one. In our experiments, we considered the password-to-password model [106] denoted as ptp. Although there are recent works proposing improved models for password reuse modeling (e.g., [158]), we did not apply them in honeyword generation in our experiments, due to the unavailability of their implementation code.
- Input-dependent, heuristically constructed models: We considered the chaffing-by-tweaking method (denoted as cbt), introduced in the original honeywords paper [69]. This method generates honeywords by randomly replacing the last several characters

of the input password. In other words, it assigns equal probability to all passwords that can be obtained by tweaking the input password.

We have introduce these password generative models in details and how we trained them in Sec. 3.1. After training PCFG, Markov, and RNN, we capped the maximum probability assigned by these models at 3×10^{-6} (i.e., $\theta_{\mathcal{G}} = 3 \times 10^{-6}$). After the training of the password-to-password model, we capped the maximum probability assigned by the password-to-password model at 10^{-8} (i.e., $\theta_{\mathcal{G}} = 10^{-8}$). In our experiments of implementing chaffing-by-tweaking model, we adapted the original method to randomly modify characters so that the number of possible candidates generated from any given password is at least $94^4 - 1$ (i.e., 78,074,895). This ensures that the maximum probability assigned by the chaffing-by-tweaking model is less than 1.3×10^{-8} (i.e., $\theta_{\mathcal{G}} = 1.3 \times 10^{-8}$). All these steps ensure that p_t^* remains sufficiently small, so that observing $Y_t = 1$ (i.e., a honeyword attempt) yields a sufficiently large increment $I_t(Y_t)$ to effectively increase the test statistic M_t.

In our experiments, we implemented our honeyword-generation algorithm using the password generative models described above. As our default, we implemented a WSWoR method with $k_i = 9$ for all users, using password generative models from the following categories: input-independent, machine-learning; input-dependent, machine-learning; and input-dependent, heuristically constructed. We also explored how the choice of k_i in WSWoR impacts the performance of our approach, in Sec. 4.4.3. For the Bernoulli-sampling-based method, we used the input-independent, heuristically constructed model (i.e., the random generator), since independently sampling passwords with heterogeneous probabilities is costly. We set the probability of independently selecting each password as honeyword to 10^{-7} in this Bernoulli-sampling-based method.

4.4.2.3 False-Alarm Attack and Breach Attack

We implemented the false-alarm attack and breach attack by using the strategies described in Sec. 4.4.1. For false-alarm attack, we randomly sampled 1,000 indices from $\{1, 2, \ldots, n\}$ without replacement as A. For breach attack, we estimated $\mathbb{P}_{\mathcal{B}_1}(x')$ using

Table 4.1: Summary of password generative models used in our experiments.

Notation	Model	Sampling	
		WSWoR	Bernoulli
pcfg	Probabilistic context-free grammar	✔	✗
mkv	Markov model	✔	✗
rnn	Recurrent neural network	✔	✗
ptp	Password-to-password model	✔	✗
cbt	Chaffing-by-tweaking model	✔	✗
rgm	Random generator	✗	✔

the dataset $D_2^{'}$ that was obtained by taking the multiset union of all multisets in D_2. Specifically, following prior work (e.g., [157]), we estimated $\mathbb{P}_{\mathcal{B}_1}(x')$ as:

$$\mathbb{P}_{\mathcal{B}_1}(x') \leftarrow \begin{cases} \frac{\mathsf{num}(x')}{|D_2^{'}|} & \text{if } x' \in D_2^{'}, \\ \frac{1}{|D_2^{'}|} & \text{if } x' \notin D_2^{'}, \end{cases}$$

where $\mathsf{num}(x')$ is the number of occurrences of x' in $D_2^{'}$. In addition, under $\mathcal{B}_2$, we define the password similarity metric sim as the cosine similarity between the embeddings of two input passwords, i.e., $\mathsf{sim}(x', x'') = \mathsf{cos}(\mathsf{embed}(x'), \mathsf{embed}(x''))$ where embed is a deep neural network that learns the embeddings of passwords and cos denotes cosine similarity. We followed the previous works (e.g., [23, 61]) to train embed on D_2, which has been introduced in Sec. 3.2.2.

4.4.2.4 Breach Detection

We implemented the breach detection algorithm by setting the inputs as $W = 2 \times 10^4$ and $\alpha = 10^{-5}$ (such that the expected annual cost due to false alarms for a site is bounded by tens of dollars, given that the average cost of breach detection and escalation in 2025 is \$1.47 million [63, p. 12]) by default. We also explored different options of W in our experiments to study its impact on the performance of our approach, in Sec. 4.4.3.

4.4.2.5 Evaluation Metrics

We used *number of compromised accounts* (NCA) and *true-detection rate* (TDR) to the evaluate the performance of our approach. NCA is the number of the accounts that a breach

attacker can access (i.e., by successfully distinguish the user passwords from honeywords) before our breach-detection algorithm detects the breach. TDR is the percentage of experiments where our breach-detection algorithm detects the breach under the breach attack. A smaller number of compromised accounts and a higher detection success rate indicate a better performance of our approach.

4.4.3 Experimental Results

4.4.3.1 Robustness Against False-Alarm Attacks

In our experiments of evaluating the performance of LeakSentinel under the false-alarm attacks (defined in Sec. 4.4.1.1), *no* false detection was triggered over 20 trials in each setting under the false-alarm attacks, which empirically confirms the bounded false-detection rate of LeakSentinel (i.e., $\alpha = 10^{-5}$). We emphasize that although, in our experiments, we configured the false-alarm attackers with the capability to compromise $|A| = 10^3$ users and perform up to $\gamma = 10^4$ login attempts per account, the provably bounded false-detection rate of LeakSentinel holds without any assumptions on the number of false-alarm attackers or their capabilities, as formally proved in Sec. 4.3.3.

Table 4.2: Empirical false-detection rates (number of detections / 20 trials) of LeakSentinel under false-alarm attacks for different $t_{\mathcal{F}}$. $t_{\mathcal{F}}$ is the time state t of the breach-detection algorithm when the false-alarm attacker started his login attempts.

$t_{\mathcal{F}}$	0	10^8	10^9	10^{10}	10^{11}
pcfg	0/20	0/20	0/20	0/20	0/20
mkv	0/20	0/20	0/20	0/20	0/20
rnn	0/20	0/20	0/20	0/20	0/20
ptp	0/20	0/20	0/20	0/20	0/20
cbt	0/20	0/20	0/20	0/20	0/20
rgm	0/20	0/20	0/20	0/20	0/20

4.4.3.2 Detecting Breach Attacks

Under naive breach attack strategies: We present the detection results of LeakSentinel under the naive breach attack strategies (defined in Sec. 4.4.1.2) in Fig. 4.1. As shown in Fig. 4.1, the changepoint ν at which the breach attackers started his login attempts had minimal im-

pact on the detection performance of LeakSentinel: a larger ν slightly delayed the detection. Under knowledge $\mathcal{B}_1$, using ptp or cbt as $\mathcal{G}$ in LeakSentinel yielded the weakest detection performance when the attacker used the likelihood greedy strategy or reward-cost-ratio greedy strategy. In these cases, a minimum of $\mathsf{NCA} \approx 1.1 \times 10^6$ (i.e., 11% of users) was required to reach a 20/20 DSR. This is because these two models generate honeywords by tweaking user passwords. Consequently, for those accounts using passwords that are popular (i.e., those having high probabilities under $\mathcal{B}_1$), the generated honeywords were less likely than the user passwords, making them easier for the breach attacker to distinguish. In contrast, under $\mathcal{B}_1$, rnn achieved the best detection performance among all password generative models considered in experiments, requiring only $\mathsf{NCA} = 1$ to reach a 20/20 DSR when the breach attacker applied the likelihood greedy strategy or reward-cost-ratio greedy strategy. LeakSentinel using pcfg or mkv also achieved strong detection performance under these settings: it required $\mathsf{NCA} \leq 152$ to reach a 20/20 DSR.

The knowledge $\mathcal{B}_2$ posed a stronger threat to LeakSentinel when it used pcfg, mkv, or rnn. Specifically, when LeakSentinel employed pcfg, it exhibited the weakest detection performance under the reward-cost-ratio-based greedy strategy, requiring a minimum of $\mathsf{NCA} \approx 2.4 \times 10^6$ (i.e., 24% of users) to achieve a 20/20 DSR. Nevertheless, this outcome remained reasonable given that approximately 43% of users in our experiments reused their passwords across sites. Using the same attack strategy, LeakSentinel with rnn needed at least $\mathsf{NCA} \approx 3.5 \times 10^4$ for a 20/20 DSR, which is 3.5×10^4 times larger than that required under $\mathcal{B}_1$. In contrast, LeakSentinel with ptp, cbt, or rgm achieved better detection performance under $\mathcal{B}_2$ than under $\mathcal{B}_1$. This is because ptp and cbt that randomly tweak the user password generate honeywords similar to the user passwords, while integrating rgm into Bernoulli sampling could produce honeywords similar to the passwords from the same users at other sites. As such, the breach attacker knowing $\mathcal{B}_2$ might mistake honeywords as user passwords and use them for login attempts, causing LeakSentinel to detect the breach after a few number (about 50 for ptp, and $\leq 10^4$ for cbt and rgm) of compromised accounts.

Under adaptive breach attack strategies: We present the detection results of LeakSentinel

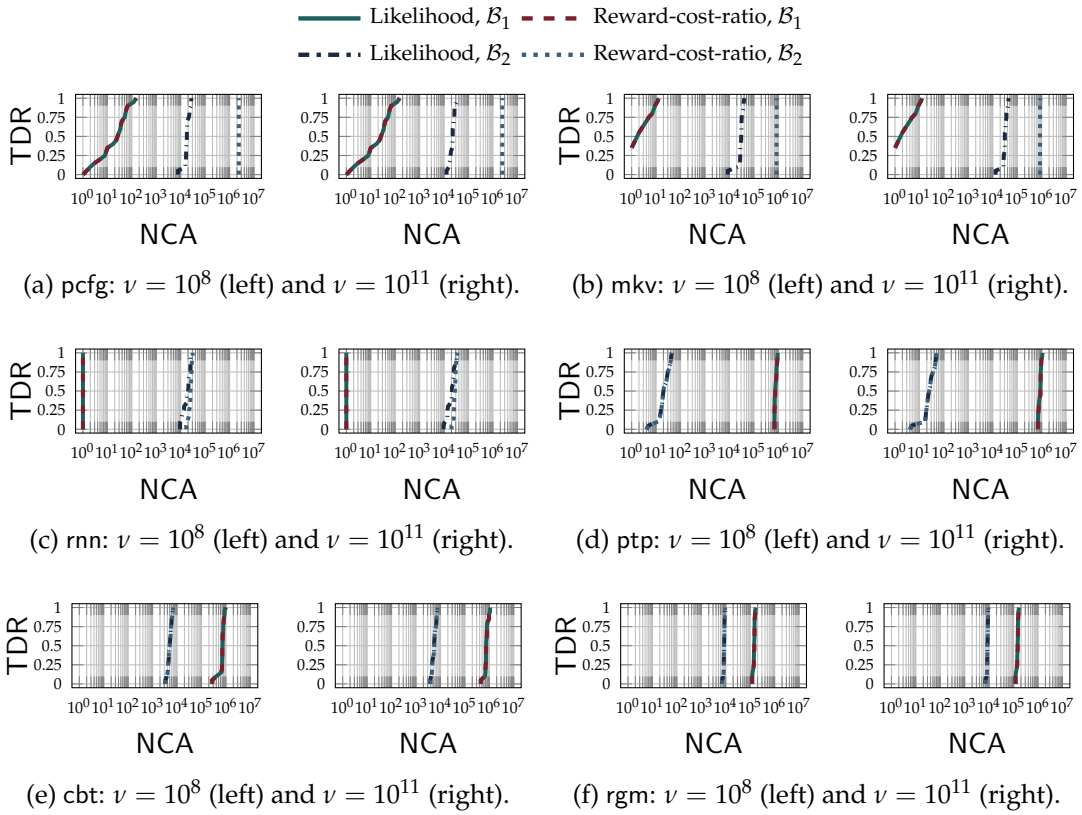
under the adaptive breach attack strategies (also defined in Sec. 4.4.1.2) in Fig. 4.2. In Fig. 4.2, we also observe that the changepoint ν when the breach attackers started his login attempts had minimal impact on the detection performance of LeakSentinel. Compared with naive breach attack, the adaptive attack is more effective at mitigating the detection of LeakSentinel, resulting a larger minimum of NCA needed for a 20/20 DSR. However, this improved evasiveness comes at the cost of a substantially higher number of login attempts that use the passwords outside the corresponding sweetword sets.

Under $\mathcal{B}_1$, LeakSentinel with ptp or cbt had the weakest detection performance under the likelihood-based strategy or the reward-cost-ratio-based strategy. Specifically, LeakSentinel using ptp yielded a minimum of NCA $\approx 2.4 \times 10^6$ (i.e., 24% of users) for a 20/20 DSR, while LeakSentinel using cbt required NCA $\approx 2.7 \times 10^6$ (i.e., 27% of users). In contrast, LeakSentinel with rnn achieved the best detection performance against adaptive attacks when the attacker was equipped with $\mathcal{B}_1$. Specifically, under both the likelihood-based greedy adaptive strategy and the reward-cost-ratio-based greedy adaptive strategy, LeakSentinel with rnn detected the breach with a 20/20 DSR after only a single compromised accounts.

Similar to the naive breach attack scenarios, $\mathcal{B}_2$ posed a strong threat to LeakSentinel using pcfg, mkv, or rnn. Specifically, when LeakSentinel used pcfg, mkv, or rnn with WSWoR, it yielded a minimum of NCA ranging from 1.9×10^6 (i.e., 19% of users) to 4.3×10^6 (i.e., 43% of users) required to reach a 20/20 DSR. Nevertheless, these outcomes remained reasonable given that approximately 43% of users in our experiments reused their passwords across sites. In contrast, LeakSentinel using ptp with WSWoR achieved the best performance in this setting: it achieved a 20/20 DSR after 65 accounts were compromised.

Takeaways: LeakSentinel instantiated with different password generative models achieved a 100% DSR once the breach attacker exploited a sufficient number of accounts, which ranged from a single account to 43% of accounts depending on the attack and model. Our evaluations showed no password generative model when employed in LeakSentinel consistently outperformed others across all breach attack strategies. Under breach attacks equipped with $\mathcal{B}_1$, LeakSentinel using an input-independent, machine-learning model (e.g., pcfg, mkv, or

rnn) generally achieved stronger detection performance than alternatives. In contrast, under breach attacks equipped with $\mathcal{B}_2$, LeakSentinel using an input-dependent model (e.g., ptp or cbt) required a smaller minimum number of NCA to reach reliable detection. These results suggest a trade-off between model types under different attacker knowledge assumptions. Designing and selecting password generative models to be used in LeakSentinel that remain optimal across diverse threat models is an important direction for future work.



(a) pcfg: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (b) mkv: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

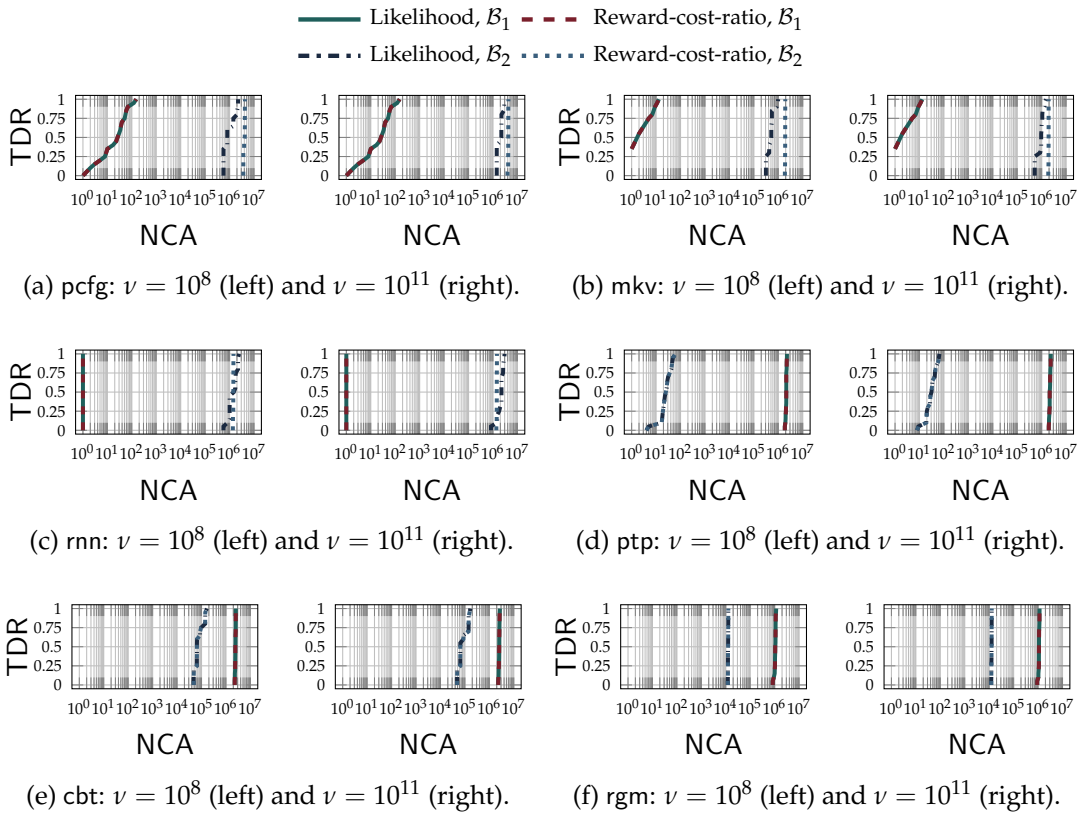
(c) rnn: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (d) ptp: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

(e) cbt: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (f) rgm: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

FIGURE 4.1: TDR of LeakSentinel under naive breach attack.

4.4.3.3 How Detection Performance Scales with n

In our experiments, we set the number n of users as 10^7 in the default setting. In this section, we study how the detection performance of LeakSentinel scales with n. We considered $n \in \{10^3, 10^4, 10^5, 10^6\}$. We report the results in Fig. 4.3 and Fig. 4.4. As shown in Fig. 4.3 and Fig. 4.4, LeakSentinel consistently achieved a 20/20 TDR under all



(a) pcfg: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (b) mkv: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

(c) rnn: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (d) ptp: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

(e) cbt: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right). (f) rgm: $\nu = 10^8$ (left) and $\nu = 10^{11}$ (right).

FIGURE 4.2: TDR of LeakSentinel under adaptive breach attack.

breach attack strategies once the breach attacker attempted to exploit a sufficiently large fraction of users, even when n was small (e.g., $n = 10^3$). For example, at a site with $n = 10^3$, LeakSentinel in all cases achieved a 20/20 TDR after more than 596 accounts (i.e., $\approx$ 60%) were compromised, when the breach attacker applied the adaptive attack strategies. Our study considering a small n also simulated scenarios in which the breach attacker compromised the credential database of a large site but recovered the plaintexts of sweetwords for only a small fraction of accounts.

4.4.3.4 The Impact of k_i in WSWoR

We study how k_i in a WSWoR honeyword-generation algorithm impacts the detection performance of LeakSentinel. In this study, we considered $k_i \in \{9, 19, 49, 99\}$. We set $n = 10^4$, $W = 2 \times 10^4$, and $\alpha = 10^{-5}$. We report the results in Fig. 4.5 and Fig. 4.6. As shown in Fig. 4.5 and Fig. 4.6, increasing k_i often reduced the minimum number of NCA

required to achieve a 20/20 DSR. However, this trend was not consistent across all settings. For example, under an adaptive breach attack using likelihood-based strategy equipped with knowledge $\mathcal{B}_1$, LeakSentinel using pcfg needed fewer compromised accounts for a 20/20 DSR when k_i was larger. In contrast, under the same attack, LeakSentinel using mkv required a larger minimum of NCA if we increased k_i. This is because setting a larger k_i makes it harder for the breach attacker to distinguish the user passwords from honeywords but it also reduces the induced increment (i.e., $I_t(Y_t)$ with $Y_t = 1$). Moreover, a larger k_i increases the storage cost. Therefore, although site could choose a large k_i to potentially reduce the number of NCA required for successful detection, we recommend setting $k_i = 9$ as a practical choice that balances detection effectiveness and storage cost.

4.4.3.5 The Impact of W

We study the impact of W on the performance of LeakSentinel of detecting breaches. In this study, we considered $n = 10^7$, and set $k_i = 9$ across users when using a WSWoR honeyword-generation algorithm and $\alpha = 10^{-5}$. Note that the choice of W does not affect the false-detection guarantee of LeakSentinel. We considered $W \in \{2 \times 10^2, 2 \times 10^3, 2 \times 10^4, 2 \times 10^5\}$ where $W = 2 \times 10^4$ is our default. We report the results in Fig. 4.7 and Fig. 4.8. As shown in Fig. 4.7 and Fig. 4.8, selecting a smaller W slightly improved the detection performance under the naive breach attack but degraded the performance under the adaptive breach attack. Conversely, setting a larger W slightly improved the detection performance under the adaptive breach attack but degraded the performance under the naive breach attack.

4.4.3.6 Summary

LeakSentinel equipped with different honeyword-generation algorithms achieved a 20/20 TDR under all breach attacks considered in our experiments, when the breach attacker attempted to exploit a sufficient fraction of accounts. At the same time, LeakSentinel guarantees a small, provably bounded global false-detection rate (e.g., 10^{-5}) under any false-alarm attacks. Moreover, the detection performance of LeakSentinel scaled well with

the number of users n. In particular, LeakSentinel remained effective at detecting breach attacks even when deployed at sites with a relatively small user base (e.g., $n = 10^3$).

4.5 Discussion

4.5.1 Integrating LeakSentinel into Alternative System Designs

LeakSentinel can be integrated into any other system design that can distinguish user passwords from honeywords at login time, such as ErsatzPasswords [5]. To integrate LeakSentinel into ErsatzPasswords, we apply the WSWoR-based honeyword-generation algorithm to generate a single honeyword for each account. In this system design, the generated honeyword per account is referred to as *ersatzpassword*. The site then applies a machine-dependent function to hash the user password such that its hash value matches that of its honeyword under the traditional hashing scheme. Subsequently, the site stores the hash values of user passwords in the credential database and runs the breach-detection algorithm to monitor the login attempts, following the approach introduced in Sec. 4.3.2. To achieve our false-detection guarantees, a mechanism that removes honeywords upon use is required. Since ErsatzPasswords generates only one honeyword per account, the system should require the user to reset her account whenever her honeyword is submitted in a login attempt, since no honeyword remains after deletion.

However, LeakSentinel cannot be extended to Lethe [37] or Amnesia [162], as these systems cannot distinguish user passwords from honeywords at login time. Our breach-detection algorithm relies on measuring whether an incorrect login attempt uses a honeyword, which requires knowing whether the submitted credential is the user password or not. To adapt LeakSentinel to system designs that lack this capability, the breach-detection algorithm could instead measure whether a login attempt uses a sweetword. Specifically, we would define: $Y = 1$ if the login attempt uses a sweetword, and $Y = 0$ if it uses a password outside the corresponding sweetword set. However, this adapted version has a critical limitation: p_t cannot be bounded, as it could equal 1 when the submitted password is the actual user password.

Designing a new honeyword system that is compatible with LeakSentinel while addressing the limitations of the existing systems, including honeychecker-based systems, Ersatz-Passwords, Lethe, and Amnesia, is beyond the scope of this work and remains an important direction for future research.

4.5.2 Password Generative Model

The honeyword-generation algorithm of LeakSentinel is compatible with any password generative model that can be applied in either a WSWoR process or a Bernoulli-sampling process, although its implementation might present some practical challenges. Therefore, any advanced password generative model proposed in the future could contribute to the improvement of LeakSentinel. A promising example of such advanced models is LLM [12, 104, 150]. To integrate an LLM in our honeyword-generation algorithm, its sampling method must be configured for weighted sampling rather than commonly-used LLM sampling techniques such as greedy search or top-k sampling. However, the computational cost of using an LLM to generate outputs for millions or billions of accounts at a site presents a significant practical consideration.

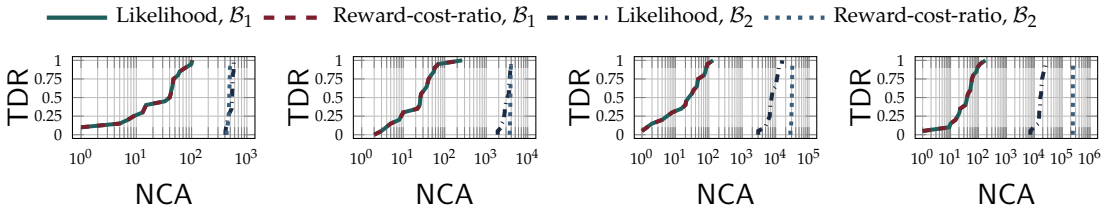
4.5.3 Breach Attacks Targeting Few Accounts

From our experimental results shown in Sec. 4.4.3, considering a site with $n = 10^7$ users, LeakSentinel required a minimum of NCA ranging from 1 to 4.3×10^6 for a 100% TDR of detecting the breach. In these settings, if a breach attacker stops his attack after compromising only a small number of accounts, LeakSentinel may fail to detect the breach. However, leaked credentials are often traded or shared through underground markets, forums, and automated services [116, 144]. As additional attackers obtain and exploit these leaked credentials to compromise accounts at the site, LeakSentinel will eventually detect the breach.

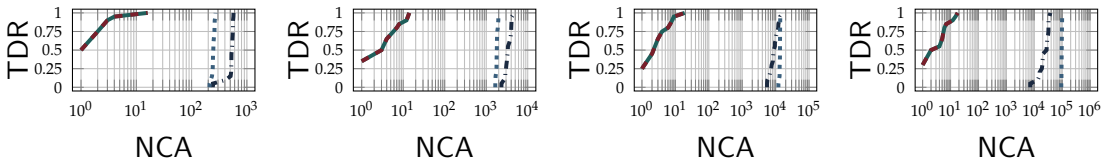
4.6 Chapter Summary

In this chapter, we proposed the first anytime-valid honeyword framework, LeakSentinel, that allows a site to detect credential-database breaches while guaranteeing a tunable, prov-

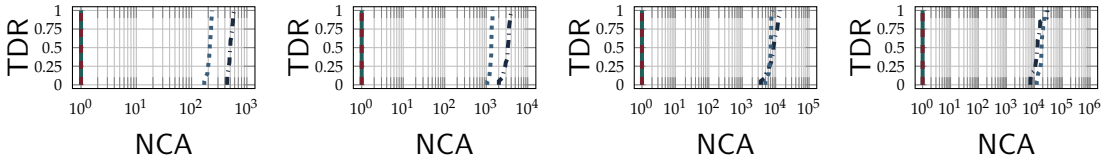
ably bounded global false-detection rate. It consists of a honeyword-generation algorithm that is compatible with an arbitrary password generative model and a breach-detection algorithm that operates online and leverages a sequential statistical test to analyze the sequence of incorrect login attempts. Through evaluations of our proposed honeyword framework, we empirically validated its false-detection guarantee and its effectiveness of detecting credential-database breaches. We thus believe that LeakSentinel provides a useful and reliable tool for a site to detect credential-database breaches.



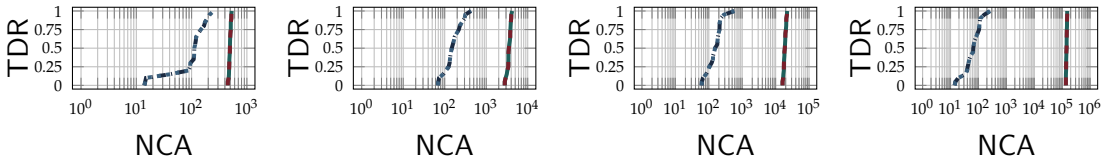
(a) pcfg: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



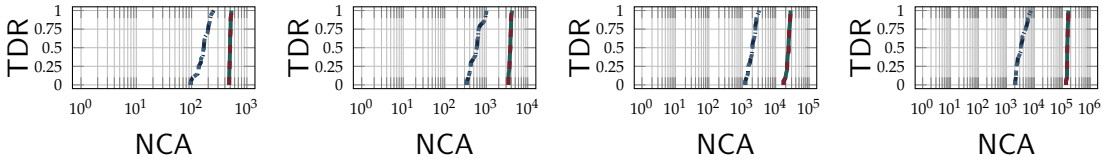
(b) mkv: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



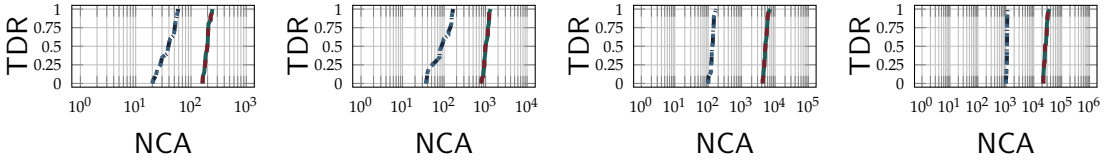
(c) rnn: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



(d) ptp: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

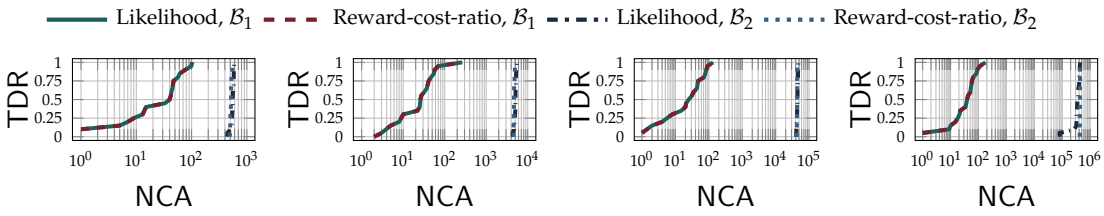


(e) cbt: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

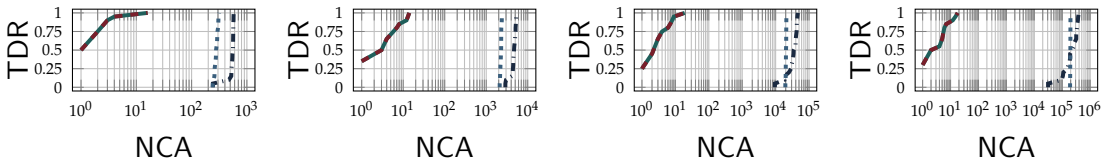


(f) rgm: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

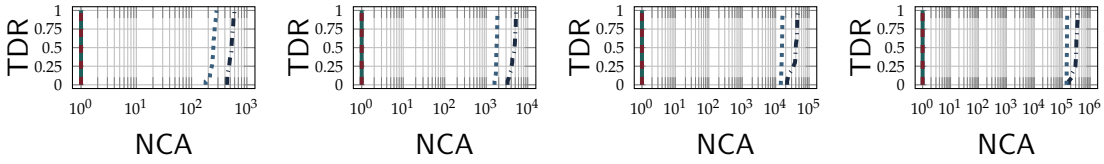
FIGURE 4.3: TDR of LeakSentinel applied by a site with different n under naive breach attack strategy.



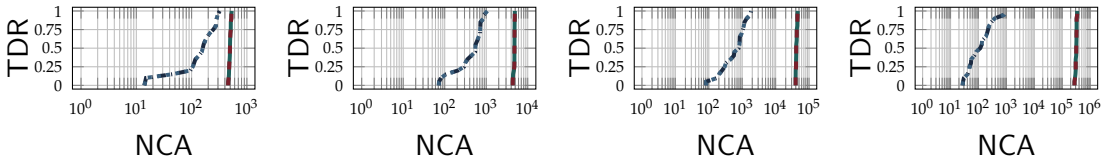
(a) pcfg: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



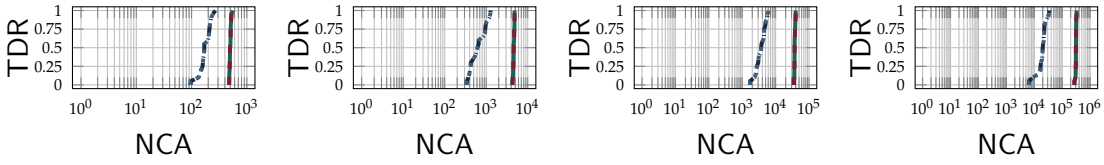
(b) mkv: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



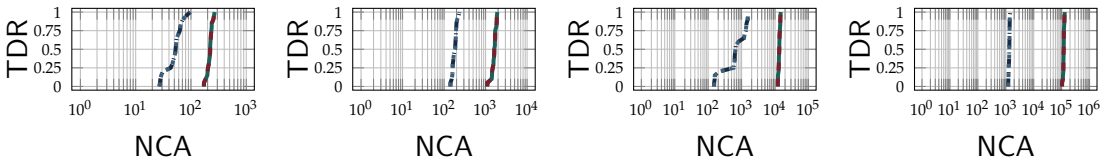
(c) rnn: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.



(d) ptp: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

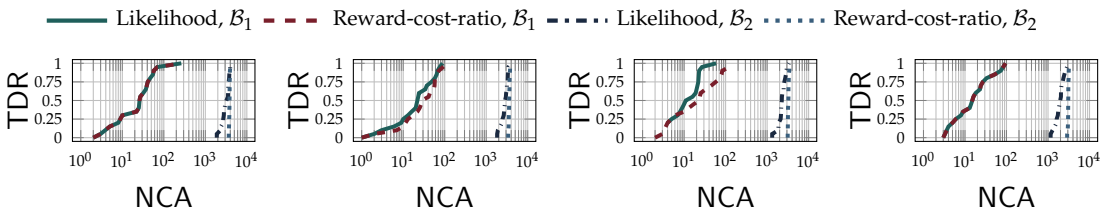


(e) cbt: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

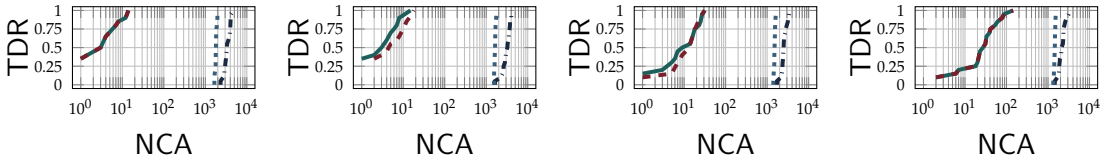


(f) rgm: From left to right: $n = 10^3$, $n = 10^4$, $n = 10^5$, and $n = 10^6$.

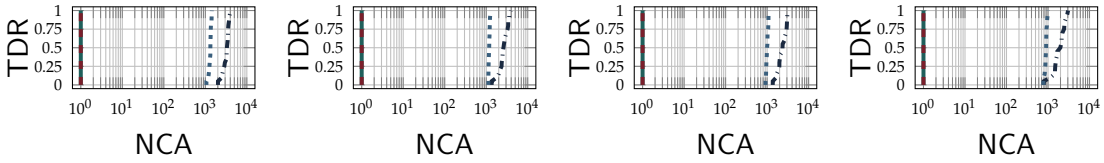
FIGURE 4.4: TDR of LeakSentinel applied at a site with different n under adaptive breach attack strategy.



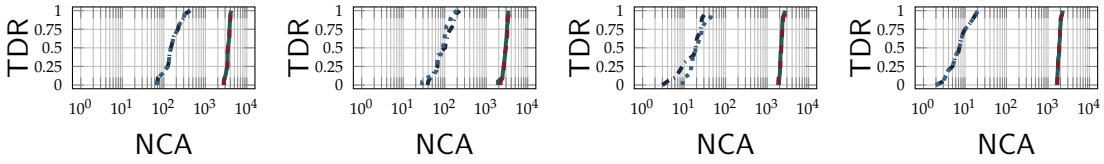
(a) pcfg: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.



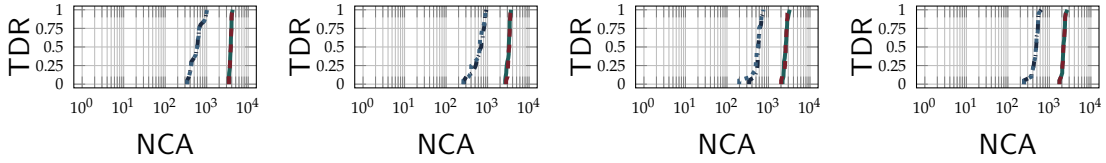
(b) mkv: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.



(c) rnn: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

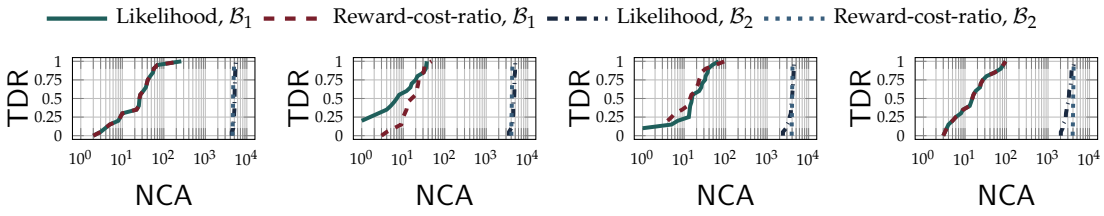


(d) ptp: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

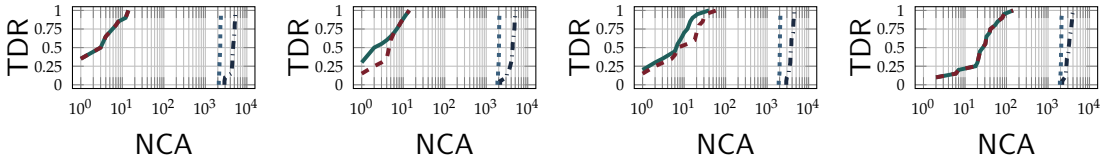


(e) cbt: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

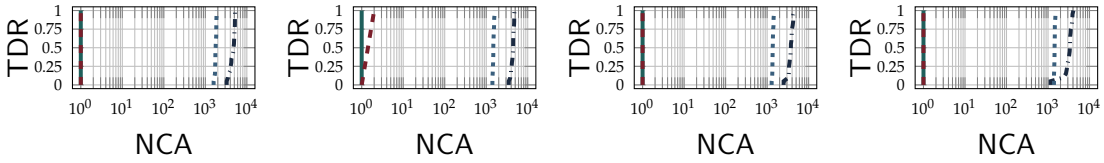
FIGURE 4.5: TDR of LeakSentinel using WSWoR honeyword-generation algorithms with different k_i ($k_i = 9$ is our default) under naive breach attack strategy.



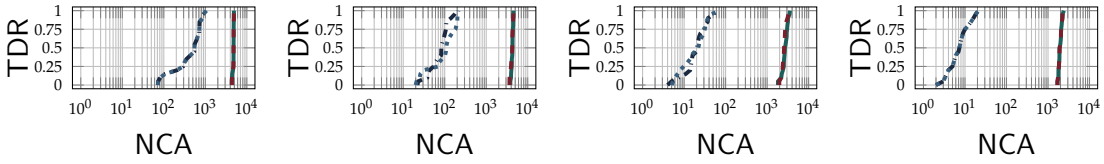
(a) pcfg: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.



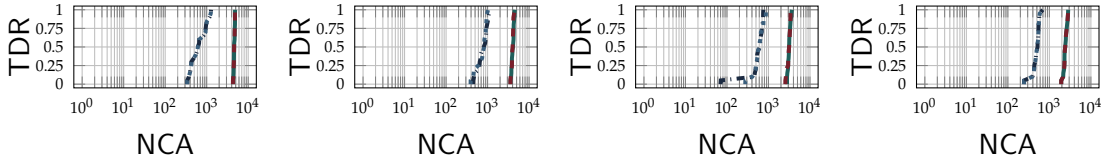
(b) mkv: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.



(c) rnn: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

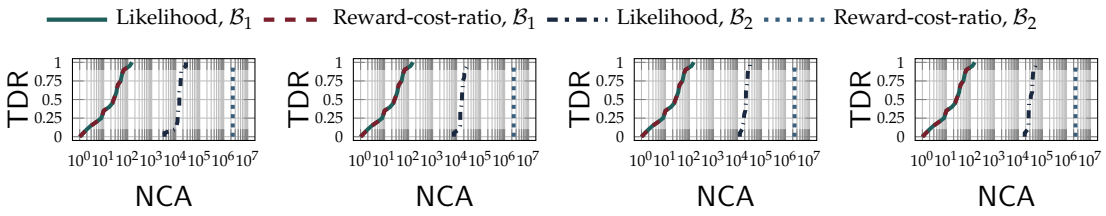


(d) ptp: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

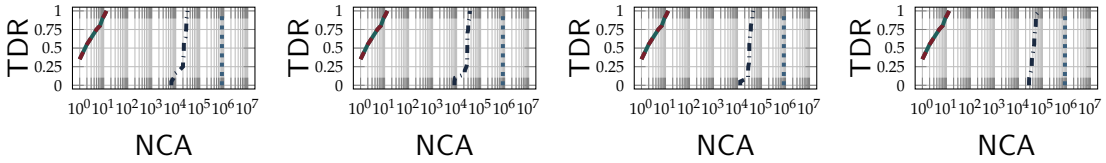


(e) cbt: From left to right: $k_i = 9$, $k_i = 19$, $k_i = 49$, and $k_i = 99$.

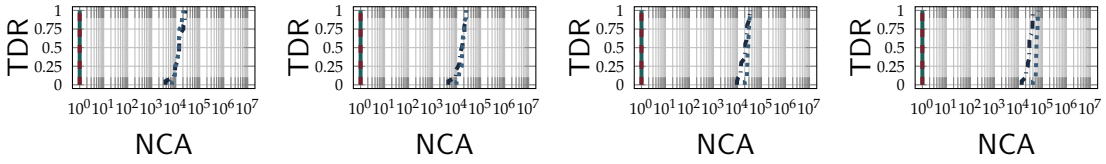
FIGURE 4.6: TDR of LeakSentinel using WSWoR honeyword-generation algorithms with different k_i ($k_i = 9$ is our default) under adaptive breach attack strategy.



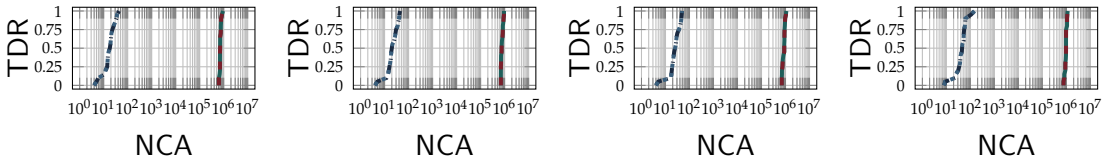
(a) pcfg: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



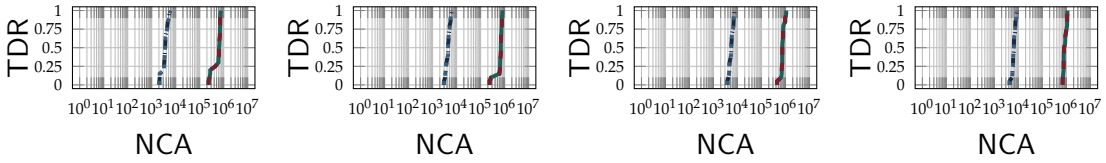
(b) mkv: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



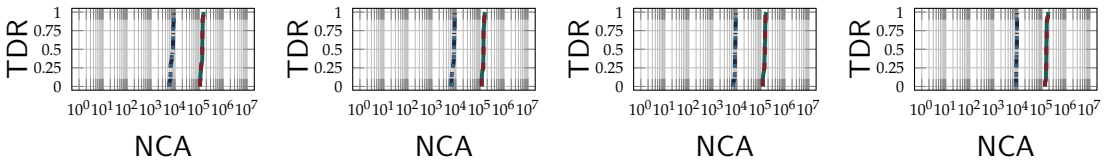
(c) rnn: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



(d) ptp: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.

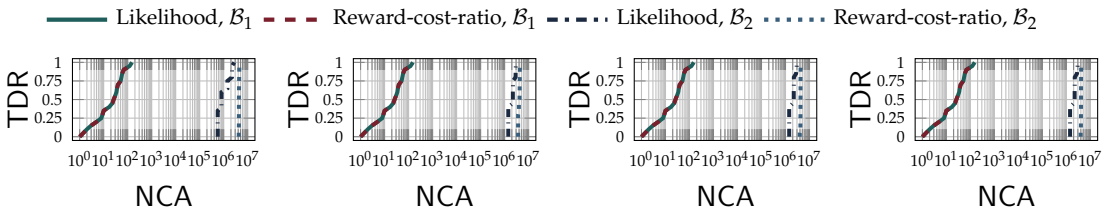


(e) cbt: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.

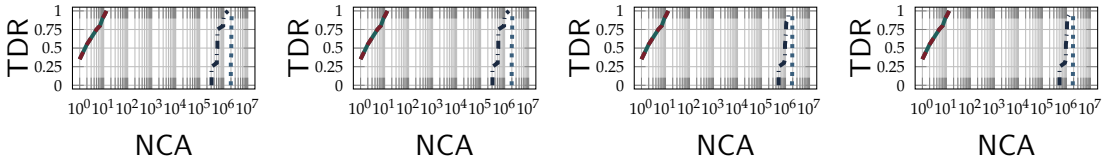


(f) rgm: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.

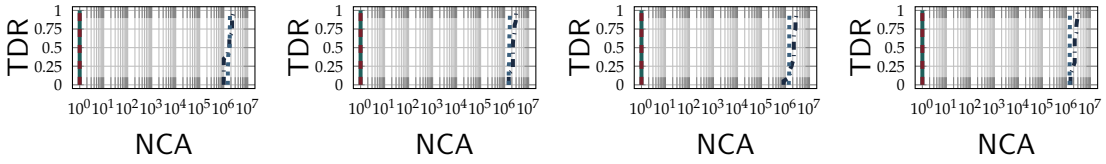
FIGURE 4.7: TDR of LeakSentinel with different W ($W = 2 \times 10^4$ is our default) under naive breach attack strategy.



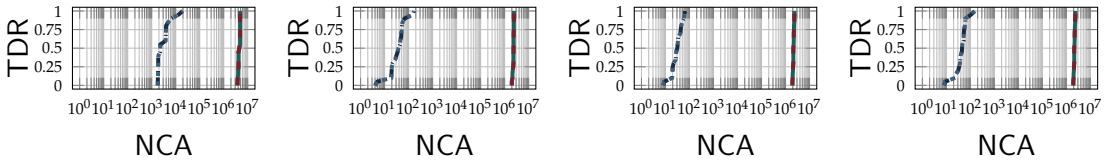
(a) pcfg: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



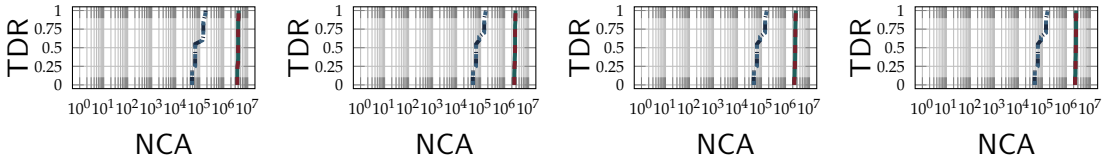
(b) mkv: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



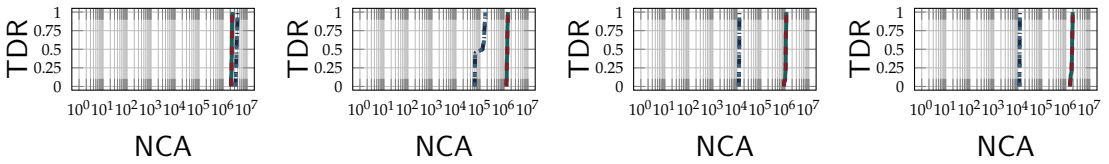
(c) rnn: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



(d) ptp: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



(e) cbt: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.



(f) rgm: From left to right: $W = 2 \times 10^2$, $W = 2 \times 10^3$, $W = 2 \times 10^4$, $W = 2 \times 10^5$.

FIGURE 4.8: TDR of LeakSentinel with different W ($W = 2 \times 10^4$ is our default) under adaptive breach attack strategy.

5. A General Framework for Data-Use Auditing of ML Models

In this chapter, we target unauthorized data use in training ML models. To address, we propose a general, proactive method that supports both dataset-level and instance-level data-use auditing of ML models. Our approach consists of two key components: a data marking algorithm and a data-use detection algorithm. The data marking algorithm, which the data owner applies prior to data publication, generates $n > 1$ distinct marked versions of a data instance by adding n unique marks (e.g., pixel alterations for images). Each marked version is carefully designed to preserve the utility of the original data instance while ensuring that the marked versions are maximally distinct. For the example of images, we could measure distinction by the distances between their high-level features prepared by a pretrained feature extractor. This marking process is agnostic to the ML task in which the marked data might be used (including, e.g., labels). After generating the n marked data, the data owner publishes only one version, selected uniformly at random, while keeping the remaining versions secret.

Once an ML model is accessible—even in only a black-box way—any "useful" membership inference method can be applied to measure the "memorization" score of each marked version, including the published one and those kept secret. If an ML model has not used the published marked data instance in training, then the rank of its "memorization" score relative to the unpublished versions should follow a uniform distribution over $\{1, 2, \ldots, n\}$ since we select it uniformly at random in the data-marking step. If, instead, the ML model has used the published marked data item in training, then its rank (based on its score) is more likely to be high, as the ML model tends to memorize its training data [134]. We develop a novel sequential method to estimate the summed ranks of the published data. This approach is anytime-valid, allowing the data owner to adaptively stop querying the audited ML model at any time while maintaining a controlled false-detection rate.

We study the performance of our data-use auditing method to audit a model for having been trained on many of a data-owner's items ("dataset level") or only a few ("instance level").

For dataset-level data-use auditing, we evaluate our method on image classifiers [34, 54, 130] and foundation models [13, 36, 114, 146]. In the first case, our results on multiple visual benchmark datasets demonstrate that our proposed framework effectively audits the use of the data-owner's data in image classifiers across various settings. Moreover, our proposed method outperforms the existing state-of-the-art dataset-level data-use auditing methods, notably Radioactive Data [118] and Untargeted Backdoor Watermark-Clean (UBW-C) [80]. We also investigate adaptive attacks that the ML practitioner might use to defeat our auditing method. While our results show that certain adaptive attacks like early stopping and differentially private stochastic gradient descent (DPSGD) can degrade the detection performance of our method, they do so at the cost of significantly diminishing the utility of the model. For the case of foundation models, we extend our evaluation to three types of foundation models: a visual encoder trained by self-supervised learning [24], Llama 2 [146], and CLIP [114]. Our results show that the proposed data auditing framework achieves highly effective performance across all of these foundation models. Overall, our proposed framework demonstrates high effectiveness and strong generalizability across different types of ML models and settings, for dataset-level data-use auditing.

For instance-level data-use auditing, which is a more challenging auditing scenario, we evaluate our method on three types of ML models, namely image classifiers [34, 54, 130], visual encoders [23], and vision-language models (Contrastive Language-Image Pretraining (CLIP) [114] and Bootstrapping Language-Image Pre-training (BLIP) [79] models). In the case of auditing image classifiers, we conducted experiments to evaluate our method on visual benchmark datasets under various settings. We empirically demonstrate the applicability of our method in scenarios where a data owner has only a few data instances (or even only one) to audit. Additionally, while the state-of-the-art passive instance-level data-use auditing methods (i.e., membership inference methods) lack formal guarantees on false-detection rates, we still performed an empirical comparison with these approaches. Our method showed comparable true-detection rates when our formal false-detection rate was set to be at the same level as the empirical rates of passive auditing methods. Our advantage

is particularly evident in practical scenarios where the data owner does not have access to reference models similar to the audited model. We also examine the effectiveness of our method against several countermeasures (e.g, preprocessing training data before their use in model training). Our findings are consistent with those observed in dataset-level auditing scenarios: while these countermeasures can degrade the detection performance of our method, they do so at the cost of substantially diminishing the utility of the audited model. Finally, we extend our evaluation to auditing visual encoders and vision-language models, further highlighting its uses across diverse ML models.

To summarize, our contributions are as follows:

- We propose a general, anytime-valid framework for auditing data use in machine learning (ML) models at both the dataset and instance levels. Our method consists of a data-marking algorithm generating n maximally distinct marked data for each raw data instance, and a detection algorithm that is built upon membership inference and a new anytime-valid statistical test that allows continuous accumulation of data-use evidence through model queries and adaptive stopping while maintaining a tunable and provably bounded false-detection rate.
- We demonstrate the effectiveness, generality, and robustness of the proposed framework by applying it to audit the use of data in diverse ML models, including image classifiers, visual encoders, LLMs, and vision-language models, in both dataset-level and instance-level settings.

Part of this chapter has been published in the 2024 ACM Conference on Computer and Communications Security [62].

5.1 Background

5.1.1 Threat Model

We consider a data owner and an ML practitioner. The data owner possesses some data instances that she intends to publish online. For instance, she may share her photos on platforms like Instagram or post text content on platforms like Twitter. The ML practitioner

aims to train a useful ML model on a training dataset for a specific task, such as developing an image classifier for classification tasks or a visual encoder to extract image features for general computer vision applications.

The ML practitioner assembles a training dataset by collecting data posted online by data owners, but does so *without their authorization*. He then trains an ML model on the collected dataset by a learning algorithm designed for his specific ML task. During training, he might apply additional techniques (e.g., image preprocessing) to reduce the likelihood that his unauthorized data use is detectable, while preserving the utility of the trained model. The model is subsequently deployed online to provide services to customers.

The goal of the data owner is to detect if the model was trained on her data instances. We have the following assumptions on the data owner's knowledge and capabilities:

- Knowledge: The data owner is unaware of the specific learning algorithm applied by the ML practitioner prior to her data publication and she does not have access to the architecture or the parameters of the deployed ML model. However, after the model is deployed, the data owner can ascertain the form of the ML model (e.g., an image classifier) and the format of its inputs and outputs (e.g., the input of an image classifier is an image while its output is a vector of confidence scores).
- Capabilities: The data owner can have a black-box access to the deployed ML model. In other words, she can obtain the output of the ML model by providing her data as input. Considering a realistic scenario, however, the data owner does not have access to a large amount of data sampled from the same distribution as the training samples used to train the deployed model. Consequently, the data owner is unable to train an ML model similar to the deployed model that could assist her in verifying whether the deployed model was trained using her data instances.

5.1.2 Data-Use Auditing in ML

Problem definition: We focus on a problem, namely *(proactive) data-use auditing in ML*, where a data owner aims to verify if a *useful* ML model deployed by an ML practitioner uses

her data in training. To make this concept precise, we define the data-use auditing of ML in a way that abstracts away the details of the system model. We do so using the experiment defined in Fig. 5.1. In data-use auditing, there are three stages, namely data marking and publication, ML model training, and data-use detection:

- Data marking and publication: A data owner has a set of data instances $X \sim \mathcal{X}$ of size $|X| = q$, where $\mathcal{X}$ represents the data distribution from which X is drawn. Prior to publishing data, the data owner applies a data-marking algorithm mark by $(X', \overline{X'}) \leftarrow \mathsf{mark}(X)$, where X' is the marked version of X that the data owner will publish and $\overline{X'}$ is the hidden information that she keeps secret and will use to detect data-use in the detection stage.
- ML model training: An ML practitioner $\mathcal{A}$ assembles his training dataset D that might include the data instances published by the data owner, i.e., $X' \subseteq D$. Then he trains an ML model f on the training dataset by $f \leftarrow \mathcal{A}(D)$.
- Data-use detection: Given oracle (black-box) access to an ML model f, the data owner applies a data-use detection algorithm detect^f by $b' \leftarrow \mathsf{detect}^f(X', \overline{X'})$, where $b' \in \{0, 1\}$. If $b' = 1$, then the data owner detects the use of her data in the ML model; otherwise, she fails to detect.

While previous works (e.g., [80, 118, 168]) address dataset-level data-use auditing where $q \gg 1$ in Fig. 5.1, our work supports both dataset-level and instance-level data-use auditing, where the data owner audits the use of an arbitrary number of data instances, including the extreme case of a single instance, in ML model training.

5.1.3 Design Goals

In this work, we aim to design a *data auditing* framework for a data owner, which she can apply to detect the ML practitioner's use of her data. We have the following design goals for the proposed data auditing framework:

- Effectiveness: The main goal of the proposed data auditing framework is to detect the unauthorized use of data in ML model training. When the published data is used,

Experiment $\mathsf{Expt}^{\text{AUDIT-}b}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m')$
  $X \sim \mathcal{X}$ such that $|X| = q$
  $(X', \overline{X'}) \leftarrow \mathsf{mark}(X)$
  $D' \sim \mathcal{X}$ such that $|D'| = m'$ and $X' \cap D' = \emptyset$
  if $b = 1$
    then $D \leftarrow D' \cup X'$
    else $D \leftarrow D'$
  $f \leftarrow \mathcal{A}(D)$
  $b' \leftarrow \mathsf{detect}^f(X', \overline{X'})$
  return b'

$$\mathsf{TDR}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') \stackrel{\text{def}}{=} \mathbb{P}\Big(\mathsf{Expt}^{\text{AUDIT-1}}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') = 1\Big)$$

$$\mathsf{FDR}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') \stackrel{\text{def}}{=} \mathbb{P}\big(\mathsf{Expt}^{\text{AUDIT-0}}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') = 1\big)$$

FIGURE 5.1: Experiment on data-use auditing in ML and measures on true-detection rate and false-detection rate.

the proposed method should successfully detect the use of the owner's data. More specifically, the detection success rate (i.e., the probability of successfully detecting the data use) should grow with the amount of the owner's data that the ML practitioner uses in training.

- Quantifiable false-detection rate: When the ML practitioner does *not* use the owner's data, then detection should occur with only a quantifiable probability (e.g., $\leq 5\%$). Such false-detection rate guarantees that if the ML practitioner does not use the data owner's data, then the risk of falsely accusing him is small and quantifiable. In other words,

$$\mathbb{P}\big(\mathsf{Expt}^{\text{AUDIT-0}}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') = 1\big) \leq \alpha,$$

  where α is a pre-defined small value, bounding the false-detection rate.

- Generality: Once the data owner publishes her data online, the ML practitioner might collect them, label them if needed, and use them in the ML-model training for his designed ML task. The generality goal is that the algorithm applied prior to data publication (i.e., the data-marking algorithm, introduced in Sec. 5.2.1) should be agnostic to the data labeling and the ML task, and that the proposed data auditing framework can be applied to effectively audit data in any type of ML model (e.g.,

image classifier or language model).

- Robustness: Once the ML practitioner realizes that the data auditing method is applied, he would presumably deploy countermeasures or adaptive attacks to defeat the data auditing method without sacrificing the utility of the trained ML model significantly. The robustness goal requires that the proposed framework is still effective to detect the unauthorized use of data in model training even when utility-preserving countermeasures or adaptive attacks have been applied by the ML practitioner.

5.2 The Proposed Method

In this section, we propose a data-use auditing method that allows a data owner to verify if her data instances $X = \{x_1, x_2, \ldots, x_q\}$ were used in training an ML model. Such a data-use auditing method consists of a data-marking algorithm mark and a data-use detection algorithm detect, which will be introduced in Sec. 5.2.1 and Sec. 5.2.2, respectively.

5.2.1 Data-Marking Algorithm

The data-marking algorithm mark, applied at the data marking and publication stage, takes as input a set of raw data instances X and outputs a marked version X' that the data owner publishes online and a set $\overline{X'}$ of hidden information that she keeps secret and will use to verify if an ML model used her published data instances in training. mark works as follows: Given a raw data instance $x_i \in X$, the data owner first generates n ($n \geq 2$) marked versions of x_i, namely $x_i^1, x_i^2, \ldots, x_i^n$. Then, she uniformly at random samples a data instance $x_i' \xleftarrow{\$} \{x_i^1, x_i^2, \ldots, x_i^n\}$ and sets $\overline{X_i'} \leftarrow \{x_i^1, x_i^2, \ldots, x_i^n\} \setminus \{x_i'\}$. As such, she publishes $X' \leftarrow \{x_1', x_2', \ldots, x_q'\}$ and keeps $\overline{X'} = \bigcup_{i=1}^{q} \overline{X_i'}$ secret.

Two requirements for n marked versions of x_i: We have two basic requirements for generating $x_i^1, x_i^2, \ldots, x_i^n$:

- *Utility preservation*: Since the published data instance x_i' should preserve the utility of the raw data instance x_i and x_i' is selected from $x_i^1, x_i^2, \ldots, x_i^n$, each marked datum x_i^j should preserve the utility of the raw data instance x_i as well. Formally, given a well-

defined distance function $\Delta_{\mathsf{uti}}(\cdot,\cdot)$ measuring the utility difference, *utility preservation* requires

$$\Delta_{\mathsf{uti}}(x_i^j, x_i) \leq \epsilon, \forall j = 1, 2, \ldots, n, \tag{5.1}$$

where ϵ is a small scalar controlling utility preservation. A smaller ϵ implies that x_i^j preserves more utility of x_i. Taking the example of images, the utility distance function could be defined as the infinity norm of the difference in the pixel values, i.e., $\Delta_{\mathsf{uti}}(x_i^j, x_i) = \left\| x_i^j - x_i \right\|_\infty$.

- *Distinction*: The n marked data should be different enough such that membership inference, applied in our data-use detection algorithm (see Sec. 5.2.2), can distinguish an ML model trained on one but not the others. Formally, given a distance function $\Delta_{\mathsf{dis}}(\cdot,\cdot)$, *distinction* requires that the minimum pairwise distance among the n marked data should be as large as possible. In other words,

$$\max_{\{x_i^1, x_i^2, \ldots, x_i^n\}} \min_{1 \leq j < j' \leq n} \Delta_{\mathsf{dis}}(x_i^j, x_i^{j'}). \tag{5.2}$$

Continuing with the example of images, we could define the distance function $\Delta_{\mathsf{dis}}(x_i, x_{i'})$ as $\Delta_{\mathsf{dis}}(x_i^j, x_i^{j'}) \stackrel{\text{def}}{=} \left\| \mathsf{Enc}(x_i^j) - \mathsf{Enc}(x_i^{j'}) \right\|_2$, where Enc is a pretrained feature extractor (e.g., pretrained on ImageNet [34]) that prepares the high-level features of an image. In this case, Eq. (5.2) is reformulated as:

$$\max_{\{x_i^1, x_i^2, \ldots, x_i^n\}} \min_{1 \leq j < j' \leq n} \left\| \mathsf{Enc}(x_i^j) - \mathsf{Enc}(x_i^{j'}) \right\|_2. \tag{5.3}$$

There exists a tension between utility preservation and distinction. Specifically, when the marked data preserves more utility of the raw data, i.e., by using a smaller ϵ, the difference between the two marked versions is smaller and thus it is harder for contrastive membership inference to distinguish between a model trained on one but not the others. In experiments in Sec. 5.3 and Sec. 5.4, we show that we can balance utility preservation and distinction well, by setting an appropriate ϵ. We also analyze and discuss this tension in Sec. 5.3 and Sec. 5.4.

Marked data generation: Given the two requirements, we formulate the problem of generating the marked data as the following optimization problem:

$$\max_{\{x_i^1, x_i^2, \ldots, x_i^n\}} \min_{1 \le j < j' \le n} \Delta_{\mathsf{dis}}(x_i^j, x_i^{j'}), \tag{5.4a}$$

$$\text{subject to} \quad \Delta_{\mathsf{uti}}(x_i^j, x_i) \le \epsilon, \forall j = 1, 2, \ldots, n, \tag{5.4b}$$

The definitions of $\Delta_{\mathsf{uti}}(\cdot, \cdot)$ and $\Delta_{\mathsf{dis}}(\cdot, \cdot)$, and how to solve Eq. (5.4) depend on the type of data. We will instantiate them in our experiments in Sec. 5.3 and Sec. 5.4.

5.2.2 Data-Use Detection Algorithm

The data-use detection algorithm is applied at the data-use detection stage after an ML model is deployed and accessible. It is used to detect if the ML model uses the published data instances in its training. Given oracle (i.e., black-box) access to a deployed ML model f, it takes as inputs a set of published data instances X' and a set of hidden information $\overline{X'}$, and outputs a bit $b' \in \{0, 1\}$ that reflects the detection result, with $b' = 1$ indicating a detection.

Obtaining "memorization" scores using any *membership inference method*: The data-use detection algorithm measures the "memorization" of each published instance x_i' compared with those marked data instances in the hidden set $\overline{X_i'}$ (i.e., the rank of x_i' among the n generated marked data according to their "memorization"). Such "memorization" can be measured by any black-box membership inference method (e.g., negative modified entropy of the output of an image classifier [137]) whose output indicates the likelihood of the input data being a training sample of an ML model (i.e., a larger output indicates a higher likelihood). Specifically, we use a membership inference method MI^f (where we are allowed black-box access to the ML model f) to obtain the "memorization" score $\mathsf{MI}^f(x_i^j)$ of a marked data instance x_i^j, which is the published data instance from X' or a marked data instance from the hidden information set $\overline{X'}$.

Formulating a rank-based hypothesis: We rank the n marked data instances of each raw data instance in increasing order with respect to their "memorization" scores, and obtain the rank

$\mathsf{rank}\big(x'_i, \overline{X'_i}, \mathsf{MI}^f\big) \in \{1, 2, \ldots, n\}$ of the published data instance x'_i where $\mathsf{rank}\big(x'_i, \overline{X'_i}, \mathsf{MI}^f\big) = n$ means that x'_i has the largest "memorization" score and $\mathsf{rank}\big(x'_i, \overline{X'_i}, \mathsf{MI}^f\big) = 1$ means it has the smallest. Specifically,

$$\mathsf{rank}\big(x'_i, \overline{X'_i}, \mathsf{MI}^f\big) = 1 + \sum_{x \in \overline{X'_i}} \mathbb{I}\big(\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)\big),$$

where $\mathbb{I}\big(\cdot\big)$ is an indicator function returning 1 if the input statement is true and 0 otherwise.[1] In other words, the rank of x'_i is the number of items in $\overline{X'_i}$ on which MI^f judges f less likely to have been trained than x'_i, plus one. In this way, we get ranks $\mathsf{rank}\big(x'_1, \overline{X'_1}, \mathsf{MI}^f\big)$, $\mathsf{rank}\big(x'_2, \overline{X'_2}, \mathsf{MI}^f\big)$, $\ldots$, $\mathsf{rank}\big(x'_q, \overline{X'_q}, \mathsf{MI}^f\big)$, and the sum of these q ranks is:

$$\mathsf{sum} = q + \xi, \tag{5.5}$$

where $\xi = \sum_{i=1}^{q} \sum_{x \in \overline{X'_i}} \mathbb{I}\big(\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)\big)$. When the ML model does not use any published data instance from X' in training—which is our null hypothesis—the rank of x'_i is uniformly distributed over $\{1, 2, \ldots, n\}$. In other words, under the null hypothesis, we have for all $i \in \{1, \ldots, q\}$ and $r \in \{1, \ldots, n\}$:

$$\mathbb{P}\Big(\mathsf{rank}\big(x'_i, \overline{X'_i}, \mathsf{MI}^f\big) = r\Big) = \frac{1}{n}.$$

Furthermore, under the null hypothesis, the distribution of the sum of ranks is that of a normalized extended binomial coefficient [14] and we have its probability mass function as:

$$\mathbb{P}(\mathsf{sum} = r) = \frac{1}{n^q} \sum_{\hat{z}=0}^{\lfloor \frac{r-q}{n} \rfloor} (-1)^{\hat{z}} \binom{q}{\hat{z}} \binom{r - n\hat{z} - 1}{q - 1},$$

where $r \in \{q, q+1, \ldots, qn\}$. However, when the ML model uses some published data instances from X' in training, it is more likely that the sum of ranks is high due to the intuition that the ML model tends to memorize its training samples [134] (i.e., some ranks

[1] We assume that $\mathsf{MI}^f(x'_i) \neq \mathsf{MI}^f(x)$. If not, we break the tie based on their indices. Specifically, if $x'_i = x'^{j}_i$, $x = x'^{j'}_i$, and $\mathsf{MI}^f(x'_i) = \mathsf{MI}^f(x)$, then we define $\mathbb{I}\big(\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)\big) = 1$ if $j > j'$, and 0 otherwise.

follow a non-uniform distribution over $\{1, 2, \ldots, n\}$). As such, we can detect the use of some published data instances from X' based on sum.

Detecting data-use sequentially: Obtaining the sum of ranks requires querying the ML model with all marked data instances, which can be highly costly when q or n is large (e.g., we consider $q = 5{,}000$ for CIFAR-100 in dataset-level auditing scenarios in Sec. 5.3 or we set $n = 1{,}000$ in instance-level auditing scenarios in Sec. 5.4). To address this, we propose a sequential method: at each time step, we sample an x from $\overline{X'}$ randomly without replacement (WoR), measure if $\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)$ (where $x \in \overline{X'_i}$), and estimate ξ and so sum (see Eq. (5.5)) using the measurements so far. Following previous works, we estimate ξ at each time step by applying a prior-posterior-ratio martingale (denoted as PPRM) [165] on the currently obtained measurements. At the time $t \in \{1, 2, \ldots, q(n-1)\}$, PPRM takes as inputs the measurements obtained so far (i.e., a sequence of t binary values, each indicating if $\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)$ where $x \in \overline{X'_i}$), the size $q(n-1)$ of $\overline{X'}$, and a confidence level $\alpha' \in [0, 1]$, and returns a confidence interval $\mathsf{CI}_t(\alpha') = [\mathsf{LB}_t(\alpha'), \mathsf{UB}_t(\alpha')]$ for ξ. Such a sequence of confidence intervals $\{\mathsf{CI}_t(\alpha')\}_{t \in \{1,2,\ldots,q(n-1)\}}$ has the following guarantee [165]:

$$\mathbb{P}\big(\exists t \in \{1, 2, \ldots, q(n-1)\} : \xi \notin \mathsf{CI}_t(\alpha')\big) \leq \alpha'.$$

In words, the probability that ξ falls outside the confidence intervals at any time step is at most α'.

As such, we design our data-use detection algorithm as follows: At each time step, she samples an x from $\overline{X'}$ randomly WoR and tests the "memorization" score $\mathsf{MI}^f(x)$. If $x \in \overline{X'_i}$ and the "memorization" score $\mathsf{MI}^f(x'_i)$ has not been measured yet, she tests the "memorization" score $\mathsf{MI}^f(x'_i)$. Then, she measures if $\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)$ where $x \in \overline{X'_i}$ and applies PPRM to obtain a confidence interval $[\mathsf{LB}_t(\alpha'), \mathsf{UB}_t(\alpha')]$ for ξ. If the lower bound $\mathsf{LB}_t(\alpha')$ of the confidence interval is equal to or larger than a preselected threshold χ, the data owner stops sampling and rejects the null hypothesis, i.e., she returns $b' = 1$ concluding that she detects the use of some members of X' in f. Otherwise, she continues sampling. When all the unpublished data instances have been exhausted and all lower bounds of confidence

intervals are smaller than the threshold χ, she returns $b' = 0$. Here χ is a tunable parameter that controls the upper bound of FDR of our method; see Theorem 7 below.

5.2.3 Guarantee on False-Detection Rate

A false detection happens when a data-use auditing method outputs $b' = 1$ when the ML model did not use any published data instance to train. FDR is the probability that false detection happens. We show that our data-use auditing method, which consists of the data-marking algorithm and data-use detection algorithm described above, allows us to set χ so that the parameter α is an upper bound on FDR. We highlight that the FDR bound is *anytime-valid*, meaning that it remains valid regardless of when the data owner stops the audit (i.e., the step of sampling an x from $\overline{X'}$ randomly WoR, testing its "memorization" score, measuring if $\mathsf{MI}^f(x'_i) > \mathsf{MI}^f(x)$ where $x \in \overline{X'_i}$, and estimating a confidence interval) in the data-use detection algorithm.

Theorem 7 (Bound on FDR)**.** *Given any $\alpha \in [0,1]$ and $\alpha' < \alpha$, when we set χ such that*

$$\sum_{r=\chi+q}^{qn} \frac{1}{n^q} \sum_{\hat{z}=0}^{\left\lfloor \frac{r-q}{n} \right\rfloor} (-1)^{\hat{z}} \binom{q}{\hat{z}} \binom{r-n\hat{z}-1}{q-1} \leq \alpha - \alpha', \tag{5.6}$$

our data-use auditing algorithm has an FDR *no larger than α. In other words:*

$$\mathbb{P}\left(\mathsf{Expt}^{\text{AUDIT-0}}_{\mathcal{X},\mathsf{mark},\mathsf{detect}}(\mathcal{A}, q, m') = 1\right) \leq \alpha.$$

Proof. We use $[q(n-1)]$ to represent $\{1, 2, \ldots, q(n-1)\}$. We study the probability that under H_0 there exists a confidence interval whose lower bound is no smaller than a prese-

lected threshold $\chi \in \{0, 1, \ldots, q(n-1)\}$, i.e., $\mathsf{LB}_t(\alpha') \geq \chi$. We have:

$$\mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid H_0\big) \tag{5.7}$$

$$= \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi \geq \chi, H_0\big) \times \mathbb{P}\big(\xi \geq \chi \mid H_0\big) \tag{5.8}$$

$$+ \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi < \chi, H_0\big) \times \mathbb{P}\big(\xi < \chi \mid H_0\big) \tag{5.9}$$

$$= \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi \geq \chi\big) \times \mathbb{P}\big(\xi \geq \chi \mid H_0\big) \tag{5.10}$$

$$+ \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi < \chi\big) \times \mathbb{P}\big(\xi < \chi \mid H_0\big) \tag{5.11}$$

$$\leq \mathbb{P}\big(\xi \geq \chi \mid H_0\big) + \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi < \chi\big), \tag{5.12}$$

where we have Eqs. (5.8)–(5.11) because the confidence sequence is independent of the null hypothesis. Under the null hypothesis, the rank of a published data instance is uniformly distributed over $\{1, 2, \ldots, n\}$ and thus the sum of independently, uniformly distributed ranks has the following probability mass function [14]:

$$\mathbb{P}(\mathsf{sum} = r) = \frac{1}{n^q} \sum_{\hat{z}=0}^{\lfloor \frac{r-q}{n} \rfloor} (-1)^{\hat{z}} \binom{q}{\hat{z}} \binom{r - n\hat{z} - 1}{q-1}.$$

Because $\mathsf{sum} = \xi + q$, we have:

$$\mathbb{P}\big(\xi \geq \chi \mid H_0\big) = \sum_{r=\chi+q}^{qn} \frac{1}{n^q} \sum_{\hat{z}=0}^{\lfloor \frac{r-q}{n} \rfloor} (-1)^{\hat{z}} \binom{q}{\hat{z}} \binom{r - n\hat{z} - 1}{q-1}.$$

Also, according to the guarantee of the confidence sequence [165], we have:

$$\mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi < \chi\big) \leq \alpha'.$$

Therefore, we have:

$$\begin{aligned}
&\mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid H_0\big) \\
&\leq \mathbb{P}\big(\xi \geq \chi \mid H_0\big) + \mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid \xi < \chi, H_0\big) \\
&\leq \mathbb{P}\big(\xi \geq \chi \mid H_0\big) + \alpha'
\end{aligned}$$

If we set χ such that

$$\mathbb{P}\big(\xi \geq \chi \mid H_0\big) \leq \alpha - \alpha'.$$

In other words,

$$\sum_{r=\chi+q}^{qn} \frac{1}{n^q} \sum_{\hat{z}=0}^{\lfloor \frac{r-q}{n} \rfloor} (-1)^{\hat{z}} \binom{q}{\hat{z}} \binom{r-n\hat{z}-1}{q-1} \leq \alpha - \alpha', \tag{5.13}$$

we have:

$$\mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid H_0\big) \leq \alpha.$$

Since our detection algorithm rejects H_0 if it finds a confidence interval whose lower bound satisfies $\mathsf{LB}_t(\alpha') \geq \chi$, $\mathbb{P}\big(\exists t \in [q(n-1)] : \mathsf{LB}_t(\alpha') \geq \chi \mid H_0\big)$ is its false-detection probability. Given any $\alpha \in [0,1]$ and $\alpha' < \alpha$, when we set χ to satisfy Eq. (5.13), our detection algorithm has an FDR no larger than α. □

5.3 Dataset-level Data-Use Auditing

In this section, we consider the scenarios in which the data owner audits a substantial amount of data instances (i.e., $q \gg 1$).

5.3.1 Auditing Image Classifiers

We apply our data-use auditing method to detect unauthorized use of data to train an image classifier. Image classification (e.g., [34, 54, 130]) is a fundamental computer-vision task in which the ML practitioner trains a model (i.e., image classifier) on training data partitioned into c classes. For a newly given image, the ML model predicts a class label for it or, more generally, a vector of c dimensions. The output vector could be a vector of confidence scores whose $\acute{z}$-th component represents the probability of the input being from the $\acute{z}$-th class, or a one-hot vector where only the component of the predicted class is 1 and the others are 0. Each training sample in the training set D is an (image, label) pair, where the image might be collected online and the label is assigned by the ML practitioner after the data collection.

5.3.1.1 Experimental Setup

Datasets: We used three image benchmarks: CIFAR-10 [72], CIFAR-100 [72], and TinyImageNet [75].

- CIFAR-10: CIFAR-10 is a dataset containing 60,000 images of $3 \times 32 \times 32$ dimensions partitioned into $c = 10$ classes. In CIFAR-10, there are 50,000 training samples and 10,000 test samples.
- CIFAR-100: CIFAR-100 is a dataset containing 60,000 images of $3 \times 32 \times 32$ dimensions partitioned into 100 classes. In CIFAR-100, there are 50,000 training samples and 10,000 test samples.
- TinyImageNet: TinyImageNet is a dataset containing images of $3 \times 64 \times 64$ dimensions partitioned into 200 classes. In TinyImageNet, there are 100,000 training samples and 10,000 validation samples that we used as test samples.

Marking setting: In each experiment, we uniformly at random sampled q samples $\{x_i\}_{i=1}^{q}$ from the training sample set of a dataset. The q samples were assumed to be owned by a data owner as X and the remaining training samples were designated as D'. Here we set $\frac{q}{q+m'} = 10\%$ as the default, i.e., $q = 5{,}000$ for CIFAR-10 or CIFAR-100, and $q = 10{,}000$ for TinyImageNet. We applied our data marking algorithm to generate the published data X' and the unpublished data $\overline{X'}$ for X. In scenarios of dataset-level data-use auditing where q is large, we set $n = 2$. In Eq. (5.4), we used $\epsilon = 10$ as the default when the pixel range of image is $[0, 255]$. We defined the marked versions x_i^1 and x_i^2 by $x_i^1 \leftarrow x_i + \delta_i$ and $x_i^2 \leftarrow x_i - \delta_i$ (δ_i is the mark), utility distance function by $\Delta_{\mathsf{uti}}(x_i^1, x_i) = \left\|x_i^1 - x_i\right\|_\infty$ and $\Delta_{\mathsf{uti}}(x_i^2, x_i) = \left\|x_i^2 - x_i\right\|_\infty$, and the distance function by $\Delta_{\mathsf{dis}}(x_i^1, x_i^2) = \left\|\mathsf{Enc}(x_i^1) - \mathsf{Enc}(x_i^2)\right\|_2$, where we used ResNet-18 [54] pretrained on ImageNet [34] to be the default feature extractor Enc. We solved Eq. (5.4) by projected gradient descent [82]. Then we uniformly at random sampled a subset (of size q') of X' as X'' (i.e., $X'' \subseteq X'$) to simulate a general case where the ML practitioner collected a subset of published data as training samples. By default, we set $q' = q$. As such, we constituted the training dataset collected by the ML practitioner as $D = D' \cup X''$ with correct labels (i.e., using the same labels as those in the dataset). We present some examples of marked images in Fig. 5.2, Fig. 5.3, and Fig. 5.4.

Training setting: We used ResNet-18 as the default architecture of the ML model f trained

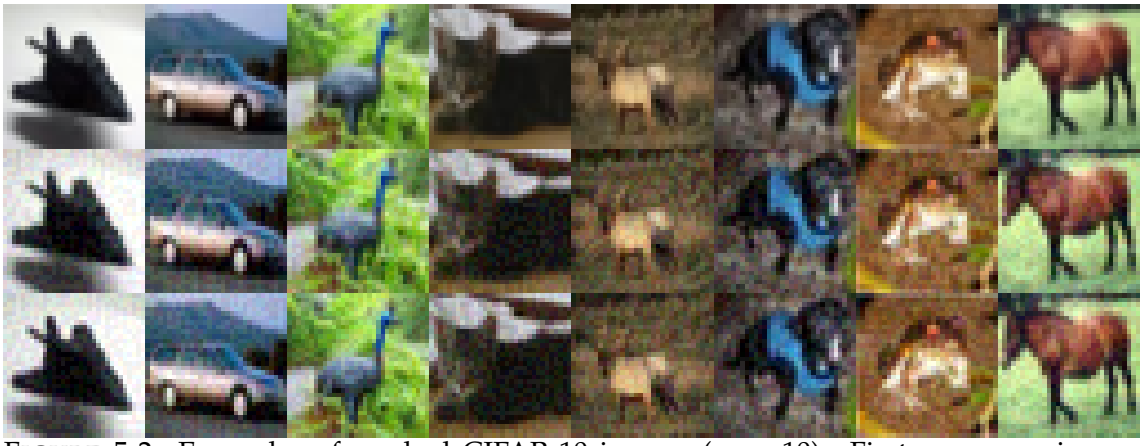


FIGURE 5.2: Examples of marked CIFAR-10 images ($\epsilon = 10$). First row: raw images; Second row: published images; Last row: unpublished images.

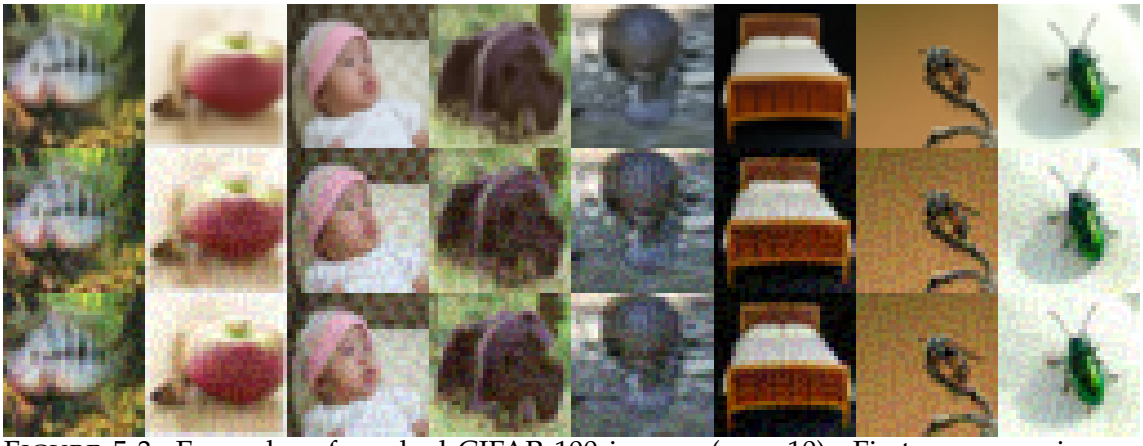


FIGURE 5.3: Examples of marked CIFAR-100 images ($\epsilon = 10$). First row: raw images; Second row: published images; Last row: unpublished images.

by the ML practitioner. We used a standard SGD algorithm to train f, as follows: f was trained on normalized training data with default data augmentation applied using an SGD optimizer [7] with a weight decay of 5×10^{-4} for 80 epochs, a batch size of 128, and an initial learning rate of 0.1 decayed by a factor of 0.1 when the number of epochs reached 30, 50, or 70.

Detection setting: In each detection experiment, we applied our data-use detection algorithm to the given ML model f using a set of pairs of generated published data and unpublished data. In the data-use detection algorithm, we followed the previous works (e.g., [27]) to define the black-box membership inference function MI^f: given an input image, we first randomly

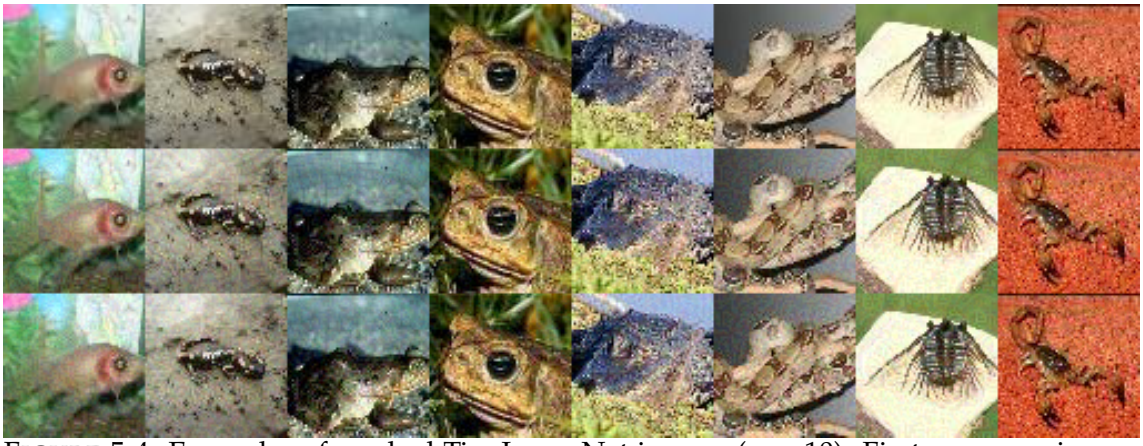


FIGURE 5.4: Examples of marked TinyImageNet images ($\epsilon = 10$). First row: raw images; Second row: published images; Last row: unpublished images.

Table 5.1: Information available to the detector. "Confidence score" indicates whether the ML model f outputs a full confidence vector ("✔") or just a label, i.e., a one-hot vector ("✗"). "Ground-truth" indicates whether the true label of a query to the ML model is known by the detector ("✔") or not ("✗").

Condition	Confidence score	Ground-truth
CG	✔	✔
C$\bar{\text{G}}$	✔	✗
$\bar{\text{C}}$G	✗	✔
$\bar{\text{C}}\bar{\text{G}}$	✗	✗

generated π perturbed versions, and then obtained π outputs using the perturbed images as inputs to the target ML model. We averaged the π outputs and use the negative (modified) entropy of the averaged output vector elements as the membership inference score. We set $\alpha' = 0.025$, $\alpha = 0.05$, and $\pi = 16$ as the default. (Recall from Thm. 7 that α bounds the false-detection rate.) We present results for four different experimental conditions that define the information available to the detector, denoted as CG, C$\bar{\text{G}}$, $\bar{\text{C}}$G, and $\bar{\text{C}}\bar{\text{G}}$. We define these four conditions in Table 5.1.

Baselines: We used two state-of-the-art methods, Radioactive Data [118], which we abbreviate to RData, and Untargeted Backdoor Watermark-Clean (UBW-C) [80], as baselines. RData requires knowledge of the class labels for its data. So, we also consider two variants of RData in which the data owner is presumed to not know how the ML practitioner will label

her data, and so applies the same mark to all of her data regardless of class ("RData (one mark)"), or to know only a "coarse" label (superclass) of the class label the ML practitioner will assign to each ("RData (superclass)"). The details of baselines and their implementation are described as follows.

- Radioactive Data (RData): RData is a dataset auditing method including a marking algorithm and a detection algorithm. The marking algorithm adds class-specific marks into a subset of the dataset. The detection algorithm tests a hypothesis on the parameters of the classifier layer of a target ML model. Prior to the marking step, the data owner needs to pretrain a feature extractor Enc, referred as the marking network, on the raw dataset; randomly samples a random unit vector $\varrho_{\acute{z}}$ for each class $\acute{z} = 1, 2, \ldots, c$; and randomly samples a subset of her dataset to be marked. In the marking step, for an image x_i of class $\acute{z}$ in the sampled subset, the marking algorithm adds a perturbation δ_i to it such that: the feature vector of $x_i + \delta_i$ (i.e., $\mathsf{Enc}(x_i + \delta_i)$) has a similar direction to $\varrho_{\acute{z}}$; the marked image is similar to its raw version in both feature space and pixel space; the magnitude of the added mark is bounded, i.e., $\|\delta_i\|_\infty \leq \epsilon$. Prior to the detection step, the data owner needs to align the feature extractor of the target classifier and the marking network by finding a linear mapping between them. After this alignment, the detection algorithm tests whether the cosine similarity between the parameter of the classifier layer associated with class $\acute{z}$ and the mark δ_i follows an incomplete beta distribution [66] (the null hypothesis) or it is large (the alternate hypothesis), and then combines the c tests to obtain a final result. If the combined p-value is smaller than 0.05, it concludes that the target classifier was trained on the data owner's marked dataset; otherwise, it was not. In the implementation of RData in our problem settings, we used their open-source code.[2] We first trained a marking network on the raw dataset by our default training method (introduced in "training setting" in Sec. 5.3.1.1). Then, we followed our marking setting (as introduced in Sec. 5.3.1.1) to constitute a marked dataset. Specifically, we randomly sampled a subset of data

[2] `https://github.com/facebookresearch/radioactive_data`

assumed to owned by a data owner and generated marked data for the raw datum from the data owner by the marking algorithm of RData, using the trained marking network and setting $\epsilon = 10$. In the application of the marking algorithm, if the labels assigned by the ML practitioner are assumed to be known, we added a mark into the data based on its label (i.e., we used the same mark for the data from the same class but different marks for different classes); if the label assignment is assumed to be unknown, we added one mark for all the data (this method is "RData (one mark)" in Sec. 5.3.1.1) or added marks based on "coarse" labels (that is "RData (superclass)"). We used the superclass of CIFAR-100 (`https://www.cs.toronto.edu/~kriz/cifar.html`) as the "coarse" labels in CIFAR-100 dataset. For the TinyImageNet dataset, we constructed "coarse" labels by querying ChatGPT (`https://chat.openai.com/`); these "coarse" labels are shown in Table 5.2. When given a target ML model, we applied its detection algorithm that returned a combined p-value. If the combined p-value was smaller than 0.05, we detected the use of marked data; otherwise, we failed.[3]

- UBW-C: UBW-C is designed based on a clean-label untargeted backdoor attack. It includes a marking (poisoning) algorithm that adds perturbations to a subset of the dataset and a detection algorithm that tests a hypothesis on the outputs of the target classifier. Prior to the marking step, UBW-C also needs to train a surrogate model on the raw dataset that is used to craft poisoned/marked data, (strategically) selects a subset of data to be marked, and selects a backdoor trigger. In the marking step, its marking algorithm crafts marked data as $x_i + \delta_i$ ($\|\delta_i\|_\infty \leq \epsilon$) such that any classifier trained on the marked data will have a desired backdoor behavior that given a test image, the classifier outputs a much lower confidence score of its ground-truth when the test image includes the preselected backdoor trigger. As such, the detection algorithm conducts a pairwise t-test [74] to test if the added backdoor trigger in the test images significantly reduces their confidence scores of their ground-truths (by comparing with a predetermined threshold τ). If the returned p-value is lower than 0.05, it concludes

[3] Setting a threshold of 0.05 guarantees that the false-detection rate is upper-bounded by 0.05.

that the classifier was trained on the marked dataset; otherwise, it was not.[4] In the implementation of UWB-C in our problem settings, we used their open-source code.[5] We trained a surrogate model used for marking data and used the same backdoor trigger as the previous work (e.g., [80]). Then, we followed our marking setting to constitute a marked dataset. Different from the setting in the previous work where the data owner has full control over the whole dataset such that she can strategically select images to be marked based on their gradient norms, our setting considers that only a subset is from a data owner. We simulated by *randomly* sampling a subset of the dataset that was assumed from the data owner. As such, we applied its marking algorithm to generate marked data for the sampled data, where we used $\epsilon = 16$ as suggested by the previous work. When given a target ML model, we applied its detection algorithm to test if the model was trained on the marked dataset based on the p-value. In the implementation of the detection algorithm, we used $\tau = 0.2$ and $\tau = 0.25$ as suggested by the previous work (e.g. [80]).

Metrics: We used the following metrics to evaluate the methods:

- Test accuracy (acc): acc is the fraction of test samples that are correctly classified by the ML model f. A higher acc indicates a better performance of the ML model.
- True-detection rate (TDR): TDR is measured in Fig. 5.1 to evaluate the effectiveness of a data-use auditing method. TDR is the fraction of experiments where a data-use auditing method (i.e., its data-use detection algorithm) returned "detected" (i.e., $b' = 1$) when the ML model was trained using X'. Under a specific bound on FDR, a higher TDR means a more effective data-use auditing method.
- Relative cost for detection (RCD): Given that the total number of *possible* queries is $q \times n \times \pi$ where q is the number of data instances audited by a data owner, n is the number of marked variants generated per data instance, and π is the number of augmentations per marked variant for query, RCD is the average percentage of that

[4] Such p-value is not the rigorous false-detection rate of UWB-C.

[5] `https://github.com/THUYimingLi/Untargeted_Backdoor_Watermark`

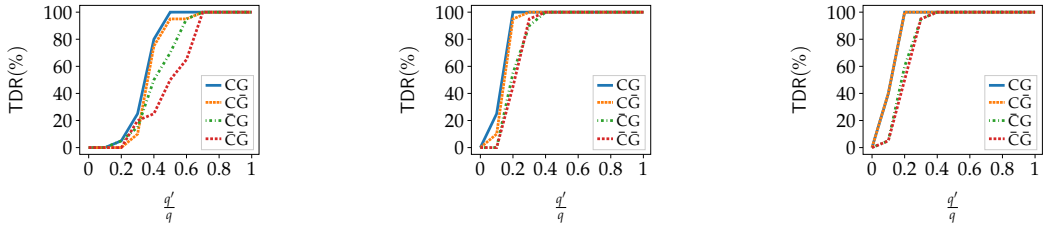
total that is queried to detect data use. It is used to measure the detection efficiency of the proposed method, which depends on the number of model queries and thus RCD.

5.3.1.2 Experimental Results on Overall Performance

Effectiveness: The detection performance of our proposed method on different visual benchmarks is shown in Table 5.3. Table 5.3 demonstrates that our method is highly effective to detect the use of published data in training ML models, i.e, yielding a 100% TDR in all settings where the published data is used as a subset of training samples of the target ML model. In addition, the ML models trained on the datasets including the published data preserved good utility, i.e., their acc values are only slightly lower ($< 1\%$ on average) than those trained on clean datasets. For detection, we needed a RCD ranging from 22.0% to 46.5% for CIFAR-10, from 1.9% to 6.0% for CIFAR-100, and from 1.4% to 6.7% for TinyImageNet. These results show that our method achieved more detection efficiency when applied to a classification task with a large number of classes. Such ranges of RCD also indicate that detection needs a number of queries to the ML model ranging from a hundred to tens of thousands. Given the current prices of online queries to pretrained visual AI models (e.g., \$1.50 per 1,000 images[6]), the detection cost is affordable, ranging from several dollars to a hundred dollars. When we have less information on the output of the ML model (i.e., the outputs are the predictions only) or the queries (i.e., the ground-truth labels are unknown) in the detection, we needed more queries to trigger detection, i.e., yielding a larger RCD.

Impact of using published data partially and false-detections: After the published data is released online, the ML practitioner might collect them partially (i.e., $\frac{q'}{q}$ is smaller than 1.0) and use the collected data in training. Here, we tested the detection performance of our method on the ML model trained on D under different ratios of $\frac{q'}{q}$. The results are shown in Fig. 5.5. When the ML practitioner used more published data, TDR was higher. Especially, when he used $\geq 70\%$ published CIFAR-10 data, or $\geq 40\%$ published CIFAR-100 data or published TinyImageNet data, we achieved a TDR of 100%, even with the least information

[6] https://cloud.google.com/vision/pricing



(a) CIFAR-10 (b) CIFAR-100 (c) TinyImageNet

FIGURE 5.5: The impact of $\frac{q'}{q}$ on the detection performance (the default $\frac{q'}{q}$ is 1.0). The results from $\frac{q'}{q} = 0$ are the false-detections of our method.

(condition $\bar{C}\bar{G}$). When the ML practitioner did not use any published data in training (i.e., $\frac{q'}{q} = 0$), TDR was 0% under all considered settings, which empirically confirms the upper bound $\alpha = 0.05$ on false-detection rate of our method.

Comparison with baselines: Table 5.4 summarizes the comparison between our method and baselines. Compared with the baselines, our method is more effective in the detection of data use, i.e., yielding a higher TDR and a higher acc. More importantly, different from the two state-of-the-art methods (i.e., RData and UBW-C), our method does not need the labeling of training samples before data publication or the white-box access to the ML model (i.e., knowing the parameters of the ML model). The variants of RData denoted as "one mark" and "superclass" do not need the complete information on labeling, but their TDR dropped significantly.

Multiple data owners: Here we consider a general real-world setting where there are multiple data owners applying data auditing independently, each of which set the upper bound on the false-detection rate as $\alpha = 0.05$. In these experiments, each data owner had 5,000 CIFAR-100 data items (i.e., 10% of the training samples collected by the ML practitioner) to publish. Each applied an auditing framework to generate her marked data and to detect its use in the deployed ML model independently. The detection results with our method and with the state-of-the-art method, RData (with full information on data labeling), are shown in Table 5.5. Compared with RData, whose detection performance degraded with a larger number of data owners, our method was much more effective, yielding a 100% TDR

in all cases.

The results in Table 5.3, Fig. 5.5, Table 5.4, and Table 5.5 demonstrate that our method achieves our *effectiveness* goal defined in Sec. 5.1.3. Table 5.4 and Table 5.5 show the advantages of our proposed method over the baselines. Table 5.5 presents interesting results under real-world settings where multiple data owners independently audit an ML model for use of their data.

5.3.1.3 Experimental Results on Impact of ML Model Architecture and Hyperparameters

In this section, we explore the impact of the ML practitioner's model architecture and the data owner's hyperparameters on detection, such as the utility bound ϵ, the feature extractor Enc used to generate marked data, the upper bound α on the false-detection rate, and the number π of sampled perturbations per image in detection.

Across different ML model architectures: Table 5.6 shows the detection results of our method on ML models across various architectures (e.g., VGG16 [130], ConvNet64 [73], ConvNetBN [65], and MobileNetV2 [122]) chosen by the ML practitioner. The results demonstrate that our method is highly effective across different ML model architectures, yielding a 100% TDR in all cases.

Impact of ϵ: Table 5.7 shows the impact of ϵ on the effectiveness of our method, demonstrating a trade-off between detection effectiveness/efficiency and data utility measured by acc of a ML model trained on it. ϵ controls the upper bound of utility difference between the published data and raw data. For images, it also controls the imperceptibility of the added marks in the published data. A smaller ϵ yields less perceptible marks (see some examples in Fig. 5.6) and allows the published data to preserve more utility of the raw data, i.e., the trained ML model has a higher acc. However, the choice of a smaller ϵ will reduce the distinction between the published data and unpublished data, and thus it slightly degrades the detection effectiveness/efficiency of our method, yielding a lower TDR (when $\epsilon = 2$) and a higher RCD.

Impact of feature extractor Enc *used in marked data generation*: Table 5.8 shows the impact

of feature extractor Enc used in the marking algorithm of our method when applied to images. Using ResNet18 [54] pretrained on ImageNet, VGG16 [130] pretrained on ImageNet, and CLIP [114] to generate marked data yielded similar detection performance. This demonstrates that our method is highly effective and efficient when a good feature extractor (e.g., pretrained on a large-scale image dataset) is used in the marking algorithm. More surprisingly, when an untrained/randomly initialized feature extractor (e.g., untrained ResNet18) was used, the efficiency of our method degraded slightly, yielding a slightly higher RCD, but it still achieved a high TDR of 100%. When there is no feature extractor available to the data owner or she does not have computing resources to craft the marked data using a feature extractor, she can craft δ_i by randomly sampling from a Gaussian distribution and clipping it such that $\|\delta_i\|_\infty \leq \epsilon$, or by randomly sampling from a Bernoulli distribution (i.e., each pixel of a mark is independently and randomly chosen to be either ϵ or $-\epsilon$ with equal probability). These variants of our method, denoted by "Gaussian mark" and "Bernoulli mark" respectively in Table 5.8, were still effective to detect the use of data but achieved a higher RCD, compared with the methods where a feature extractor was used.

Different choices of upper bound α on false-detection rate: Table 5.9 shows the performance of our data auditing method under different upper bounds on the false-detection rate, i.e., α. Under all choices of α, our method achieved a TDR of 100%. When we set a smaller α, we can control the false-detection rate to a smaller level but require more queries to detect the use of the published data in model training.

Impact of π: Table 5.10 shows the impact of π on the detection performance of our method. As shown in Table 5.10, increasing π led to a smaller RCD but a larger cost (since $\mathsf{cost} = 2 \times \pi \times G$). Therefore, setting a larger π improves the detection efficiency but degrades the cost efficiency of our proposed method.

5.3.1.4 Experimental Results on Robustness to Countermeasures

When the ML practitioner knows that a data owner marked her data, he might utilize countermeasures to defeat the auditing method. His goal is to decrease TDR without de-

grading the performance of the trained ML model significantly. We evaluated the robustness of the proposed method to three types of countermeasures, described below.

Limiting the information from the ML model output: Since our detection method measures the difference between outputs of the ML model on the published data and unpublished data, the ML practitioner can limit the output (e.g., the vector of confidence scores) of the deployed ML model, aiming to degrade our detection. Here we considered two countermeasures of this type:

- Outputting only the top κ confidence scores (Top-κ): This countermeasure allows the deployed ML model to output the top κ confidence scores, masking out the others in the output vector. Here we considered $\kappa = 1$ and $\kappa = 5$.
- Adding perturbation into the output (MemGuard [67]): This countermeasure adds carefully crafted perturbations into the ML model output to limit the information given. We considered MemGuard proposed by Jia et al. [67] to design the perturbation, where we used a moderate distortion budget of 0.5.

Note that these countermeasures can be applied only in the ML model deployment where the output is a vector of confidence scores instead of a prediction/label.

Reducing memorization of training samples: Intuitively, the ML practitioner can apply methods to discourage memorization of training samples by the ML model, so that the published data and the unpublished data will have similar scores by the defined score function. As such, reducing memorization of training samples could render the detection method to be less effective. We considered three such countermeasures:

- Differentially private stochastic gradient descent (DPSGD [3, 58]): DPSGD is the state-of-the-art private learning algorithm reducing the memorization and privacy leakage of training samples, and thus it can be considered as an attack to a data-use auditing method. It works by clipping the norm of the gradients and adding Gaussian noise parameterized by a standard deviation (i.e., noise multiplier) σ into gradients during training [3, 58].
- Early stopping (EarlyStop): In this countermeasure, the ML practitioner trains the

ML model for a small number of epochs to prevent the ML model from overfitting to the training samples. Here we trained ML models for 20, 40, and 60 epochs, denoted as EarlyStop(20), EarlyStop(40), and EarlyStop(60), respectively.

- Adversarial regularization (AdvReg [102]): Adversarial regularization is a strategy to generalize the ML model. It does so by alternating between training the ML model to minimize the classification loss and training it to maximize the gain of a membership inference attack. In the implementation of AdvReg, we set the adversarial regularization factor to be 1.0 [102].

Other attacks: We also considered some other adaptive attacks that aim to defeat our auditing method:

- Detecting pairs of published data and unpublished data in queries (PairDetect): The intuition behind this pair detection is that if the deployment can detect queries of a pair of published data and unpublished data, then it will return the same output to evade detection. We design such a pair detection method as follows: we maintain a window of queries in the history and their ML model outputs, and we compare each new query with those in the window to decide what to output. If the infinity norm of the pixel difference between the new query and a previous query is smaller than 2ϵ, we return the output of the previous query; otherwise, we return the output for the new query.
- Adding Gaussian noise into the training samples (Rand(σ)): This method adds noise into each training sample to mask the added mark. The added noise is sampled from a Gaussian distribution with standard deviation σ.
- Avoiding data augmentation in training (NoTrainAug): Excluding data augmentation in ML model training will degrade the effectiveness of the label-only membership inference that we apply as the score function for the image classifier, as demonstrated by previous works (e.g., [27]).
- Using our marking algorithm (with the default hyperparameters) to perturb training samples (MarkPerturb): This countermeasure applies our marking algorithm (with the

default hyperparameters) to craft two perturbed versions of each training sample. Since both perturbed versions remain close to the published data, including both in training would increase memorization of the published data and consequently increase the likelihood of detecting its use. Therefore, we randomly select one of the two versions for training.

Results: We summarize the robustness of our method to these countermeasures in Table 5.11. As shown in Table 5.11, masking (Top-5, Top-1, MemGuard) had limited impact on our detection effectiveness, yielding a slightly higher RCD but not changing TDR at all. DPSGD and EarlyStop did decrease TDR. However, these countermeasures damaged the utility of the trained ML model, yielding a much smaller acc. Specifically, the application of differential privacy needed a high level of privacy guarantee to defeat our method and so added a large amount of Gaussian noise into the training process to do so. The added noise affected the performance of the ML model, decreasing acc to 64.11%, more than 10 percentage points lower than acc with the default training method. Likewise, to degrade the detection performance of our method, early stopping needed to stop the training when reaching a small number of training epochs, at the cost of low accuracy as well, e.g., acc = 67.10% at 20 epochs. Among the other attacks, detecting queried pairs and excluding data augmentation in ML model training were not useful to counter our method. Pair detection (PairDetect) did not work well to detect queried pairs because we only queried the ML model with their randomly cropped versions, which evaded pair detection. Excluding data augmentation in training did not reduce TDR but diminished the accuracy of the ML model significantly, yielding a low acc of 61.59%. Adding sufficient Gaussian noise to mask the marks before training reduced the detection effectiveness of our method but, again, it also destroyed the utility of the ML model. For example, adding Gaussian noise with $\sigma = 30$ into marked CIFAR-100 data reduced TDR from 100% to 30% in condition $\bar{\text{C}}\bar{\text{G}}$ but also decreased acc to 62.10%. The last adaptive attack, i.e., applying our marking algorithm to add perturbations, did not decrease TDR but increased RCD, at the cost of achieving a lower acc of 70.49%. This is because the perturbed published data created by the marking algorithm was still

closer to the published data than to the unpublished data, which caused the published data to appear more likely to have been used in the training of the ML model trained on the perturbed published data.

In summary, countermeasures we considered in this work did not defeat our auditing method or did so at the cost of sacrificing the utility of the trained ML model; i.e., none achieved a low TDR and a high acc at the same time. Therefore, we conclude that our method achieves the *robustness* goal defined in Sec. 5.1.3 for image classifiers.

5.3.2 Auditing Visual Encoders

In this section, we apply our data auditing method to detect unauthorized use of data in visual encoders trained by self-supervised learning (e.g., SimCLR [23]). Visual encoder is a type of foundation model used to learn the general representations of images. A visual encoder can be used as a feature extractor to extract features of images in many vision recognition tasks, e.g., image classification and object detection. A visual encoder is an ML model that takes as input an image and outputs its representation as a feature vector. It is trained by self-supervised learning (e.g., SimCLR [23]) on unlabeled data (i.e., each instance in D is an image).

5.3.2.1 Experimental Setup

Datasets: We used three image benchmark datasets: CIFAR-10, CIFAR-100, and TinyImageNet.

Marking setting: We followed the setup introduced in Sec. 5.3.1.1 to generate the marked dataset, without labels needed. Our using the same marking setup indicates that the application of our marking algorithm is agnostic to the ML task.

Training setting: We followed the previous work (e.g., [23]) to train the ML model by SimCLR, which takes as inputs a base encoder and a projection head (i.e., a multilayer perceptron with one hidden layer). We used ResNet-18 as the default architecture of the base encoder. The SimCLR algorithm works as follows: at each training step, we randomly sampled a mini-batch (i.e., of size 512) of images from the training set and generated two

augmented images from each sampled instance by random cropping and resizing, random color distortion, and random Gaussian blur. The parameters of the base encoder and the projection head were updated by minimizing the NT-Xent loss among the generated augmented images, i.e., maximizing the cosine similarity between any positive pair (i.e., two augmented images generated from the same sampled instance) and minimizing the cosine similarity between any negative pair (i.e., two augmented images generated from different sampled instances). We used SGD with Nesterov Momentum [139] of 0.9 and a weight decay of 10^{-6} as the optimizer, and applied a cosine annealing schedule [86] to update the learning rate, which was set to 0.6 initially. We trained the base encoder and the projection head by $1{,}000$ epochs as the default, and returned the base encoder as the visual encoder f deployed by the ML practitioner.

Detection setting: In the detection algorithm, we followed the previous works (e.g., [83]) to define the membership inference function. The intuition behind this membership inference method is that the visual encoder f generates more similar feature vectors of two perturbed versions of a training sample than of a non-training sample [83]. In other words, if x_i^j was used in training f while $x_i^{j'}$ was not, then $\mathsf{cosim}(f(x_{i,1}^j), f(x_{i,2}^j)) > \mathsf{cosim}(f(x_{i,1}^{j'}), f(x_{i,2}^{j'}))$ where cosim denotes the cosine similarity, $x_{i,1}^j$ and $x_{i,2}^j$ are two perturbed versions of x_i^j, and $x_{i,1}^{j'}$ and $x_{i,2}^{j'}$ are two perturbed versions of $x_i^{j'}$. As such, we defined the membership inference function MI^f as follows: given an input image, we first randomly generate π of its perturbed versions (e.g., by random cropping and flipping), and then obtain π feature vectors using the perturbed images as inputs to the target visual encoder; second, we compute the cosine similarity of every pairs of feature vectors and return the sum of cosine similarities as the membership inference score. We set $\alpha' = 0.025$, $\alpha = 0.05$, and $\pi = 64$ as the default.

Metrics: We used the following metrics for evaluation:

- Test accuracy of downstream classifier (acc): acc is the fraction of test samples that are correctly classified by a downstream classifier that uses the visual encoder as the backbone and is fine-tuned on a small set of data. We followed previous work (e.g., [23])

to fine-tune the downstream classifier on the clean training samples with their labels. A higher acc indicates a better performance of the visual encoder.

- True-detection rate (TDR): Please see its description in Sec. 5.3.1.1.
- Relative cost for detection (RCD): Please see its description in Sec. 5.3.1.1.

5.3.2.2 Experimental Results

The overall experimental results on three visual benchmarks are presented in Table 5.12. As shown in Table 5.12, our proposed method achieved highly effective detection performance on auditing data in visual encoders, yielding a 95% TDR for CIFAR-10 and a 100% TDR for CIFAR-100 and TinyImageNet.

We investigated the impact of training epochs of visual encoder on the detection performance of the auditing method. We trained visual encoders on marked CIFAR-100 by epochs of 200, 400, 600, 800, and 1,000 (1,000 is the default number of epochs). As shown in Fig. 5.7, when we trained the encoder with a smaller number of epochs, the encoder memorized the training samples less and thus we had a lower TDR. However, training encoder with fewer epochs yielded modestly lower encoder utility, measured by the test accuracy of the downstream classifier (i.e., acc). This suggests that early stopping (i.e., training with a small number of epochs) can degrade the detection performance of our method, but cannot completely alleviate the trade-off between evading detection and encoder utility.

5.3.3 Auditing Llama 2

In this section, we study the application of data auditing to a large language model (LLM). An LLM is a type of large ML model that can understand and generate human language. Here we consider Llama 2 [146] published by Meta AI in 2023, which is an open-sourced LLM with notable performance and, more importantly, is free for research [146]. Specifically, Llama 2 is a family of autoregressive models that generate text by predicting the next token based on the previous ones. They are designed with a transformer architecture [150] with parameters ranging from 7 billion to 70 billion, and pretrained and fine-tuned on massive text datasets containing trillions of tokens collected from public sources [146].

It is challenging to conduct lab-level experiments on auditing data in a pretrained Llama 2 because pretraining Llama 2 on a massive text corpus needs a huge amount of computing resources. Therefore, instead of applying our data auditing method to the pretrained Llama 2, we mainly focus on a Llama 2 fine-tuning setting.

5.3.3.1 Experimental Setup

Datasets: We used three text datasets: SST2 [131], AG's news [182], and TweetEval (emoji) [100].

- SST2: SST2 is a dataset containing sentences used for sentiment analysis (i.e., there are 2 classes, "Negative" and "Positive"). In SST2, there are 67,300 training samples and 872 validation samples that we used for testing.
- AG's news: AG's news is a dataset containing sentences partitioned into 4 classes, "World", "Sports", "Business", and "Sci/Tech". In AG's news, there are 120,000 training samples and 7,600 test samples.
- TweetEval (emoji): TweetEval (emoji) is a dataset containing sentences partitioned into 20 classes. In TweetEval (emoji), there are 100,000 training samples and 50,000 test samples.

Marking setting: In each experiment, we uniformly at random sampled a subset of training samples of a dataset (e.g., 10,000 training samples). From these training samples, we uniformly at random sampled $q = 1{,}000$ sentences $\{x_i\}_{i=1}^{q}$ assumed to be owned by a data owner as X and designated the remaining samples as D'. We applied our data marking algorithm to generate the published data X' and the unpublished data $\overline{X'}$ for X. In Eq. (5.4), we defined the distance function by Levenshtein distance [77] and the utility difference function by semantic dissimilarity [36]. Instead of solving Eq. (5.4) exactly, we approximated it by using a paraphraser model (e.g., [153]) to generate two semantically similar but distinct sentences. As such, we constituted the training dataset collected by the ML practitioner as $D = D' \cup X'$, labeled correctly (i.e., using their original labels).

Fine-tuning setting: We used `Llama-2-7b-chat-hf` Llama 2 model released in Hugging Face[7] as the base model. We used QLoRA [35] to fine-tune the Llama 2 model on the marked dataset, where we applied AdamW [87] as the optimizer that was also used to pretrain Llama 2 by Meta AI [146]. We fine-tuned the model with a learning rate of 2×10^{-4}. The fine-tuned Llama 2 is the ML model deployed by the ML practitioner.

Detection setting: In the detection algorithm, We used the negative loss, a simple and effective membership inference metric [17], to measure the membership inference score of a data sample. Formally, given a text sample x_i^j that can be tokenized by a tokenizer as a sequence of integer-valued tokens $a_i^1 a_i^2 \dots a_i^l$ where each $a_i^{\bar{z}} \in \{1, 2, \dots, v^*\}$, we have the training loss of f on x_i^j as:

$$\ell(f, x_i^j) = \sum_{\bar{z}=1}^{l} -\log \left[f(a_i^1 \dots a_i^{\bar{z}-1}) \right]_{a_i^{\bar{z}}}, \tag{5.14}$$

where $f(a_i^1 \dots a_i^{\bar{z}-1})$ denotes the predicted v^*-dimensional probability vector over the token vocabulary $\{1, 2, \dots, v^*\}$, and $\left[f(a_i^1 \dots a_i^{\bar{z}-1}) \right]_{a_i^{\bar{z}}}$ denotes the $a_i^{\bar{z}}$-th component of vector $f(a_i^1 \dots a_i^{\bar{z}-1})$. We defined $\mathsf{MI}^f(x_i^j) = -\ell(f, x_i^j)$. We set $\alpha' = 0.025$ and $\alpha = 0.05$.

Metrics: We used the following metrics to evaluate methods:

- Test accuracy (acc): acc is the fraction of test samples that were correctly classified by the fine-tuned Llama 2. A higher acc indicates a better performance of the fine-tuned model.
- True-detection rate (TDR): Please see its description in Sec. 5.3.1.1.
- Relative cost for detection (RCD): Please see its description in Sec. 5.3.1.1.

5.3.3.2 Experimental Results

The results on applying our auditing method to the fine-tuned Llama 2 on three marked datasets are presented in Table 5.13. As shown in Table 5.13, when we tested the detection method on the pretrained Llama 2 (i.e., in the row of "Epoch 0"), we obtained a TDR of

[7] https://huggingface.co/meta-llama/Llama-2-7b-chat-hf

0%, indicating that the Llama 2 is not pretrained on the published data. If true, this result empirically confirms the bounded false-detection rate of our method. When we fine-tuned Llama 2 on the marked datasets by only 1 epoch, the accuracy of the fine-tuned Llama 2 model increased from 63.07% to 95.33% for SST2, from 28.41% to 91.69% for AG's news, and from 16.58% to 40.49% for TweetEval. At the same time, our method achieved a TDR of 100% on the fine-tuned model, which demonstrates the effectiveness of our method. Fine-tuning Llama 2 with more epochs increased the accuracy slightly (e.g., from 95.33% to 95.56% for SST2, from 91.69% to 92.33% for AG's news, and from 40.49% to 43.03% for TweetEval) but leads to a much lower RCD. This is because fine-tuning the model for more epochs memorizes the fine-tuning samples more and the detection method needs fewer queries to the model to detect their use.

5.3.4 Auditing CLIP

In this section, we apply data auditing to a multimodal model [114, 115]. A multimodal model is a type of ML model that can understand and process various types of data, e.g., image, text, and audio. We considered Contrastive Language-Image Pretraining (CLIP) [114], developed by OpenAI in 2021, as our study case. CLIP is a vision-language model consisting of a visual encoder and a text encoder used to extract the features of the input image and text, respectively. It takes as inputs an image and a text and returns their corresponding feature vectors. CLIP is known for its notable performance in image-text similarity and zero-shot image classification [114].

The CLIP model released by OpenAI was pretrained on 400 million image and text pairs collected from the Internet [114]. While it is challenging to pretrain such a large model on a huge number of pairs using lab-level computing resources, we aim to fine-tune the CLIP on a small (marked) dataset and test our auditing method on the fine-tuned CLIP.

5.3.4.1 Experimental Setup

Datasets: We used the Flickr30k [174] dataset. Flickr30k comprises 31,014 images of varying sizes, each paired with multiple textual descriptions.

Marking setting: In each experiment, we uniformly at random sampled q images with captions from training samples, assumed to be owned by a data owner as X, and designated the remaining training samples as D'. We set $\frac{q}{q+m'} = 10\%$, i.e., $q = 2{,}500$. We applied our data marking algorithm to generate the published data X' and the unpublished data $\overline{X'}$ for X. In the marking algorithm, given a raw datum (i.e., an image with its caption), we followed the marking setting in Sec. 5.3.1.1 to generate two marked images and then randomly sampled one with its original caption as the published data, keeping the other as the unpublished data. As such, we constituted the training dataset collected by the ML practitioner as $D = D' \cup X'$.

Fine-tuning setting: We used the CLIP model released by OpenAI as the base model.[8] We fine-tuned the CLIP model on the marked dataset D, following the pretraining algorithm used by OpenAI [114]. We used a batch size of 256 and applied Adam [70] with a learning rate of 10^{-5} as the optimizer. The fine-tuned CLIP including the visual encoder and text encoder is the ML model deployed by the ML practitioner.

Detection setting: We defined MI^f by a recently proposed membership inference on CLIP [71]. It uses cosine similarity between the two feature vectors returned by the CLIP model as the inference metric [71]. Formally, given an image-text sample $x_i^j = (\dot{x}_i^j, \ddot{x}_i^j)$, we have $\mathsf{MI}^f(x_i^j) = \mathsf{cosim}(f'(\dot{x}_i^j), f''(\ddot{x}_i^j))$, where f' and f'' are the visual encoder and text encoder of f, and cosim denotes cosine similarity. In the detection algorithm, we set $\alpha' = 0.025$ and $\alpha = 0.05$.

Metrics: We used the following metrics for evaluation:

- Test accuracy (acc): We randomly divided the test samples into batches (each is 256 at most). For each batch, we measured the fraction of texts correctly matched to images and the fraction of images correctly matched to texts, by the (fine-tuned) CLIP model. We used the fraction of correct matching averaged over batches as the test accuracy acc.

[8] https://github.com/openai/CLIP

- True-detection rate (TDR): Please see its description in Sec. 5.3.1.1.
- Relative cost for detection (RCD): Please see its description in Sec. 5.3.1.1.

5.3.4.2 Experimental Results

The overall performance of our data auditing method applied in fine-tuned CLIP is presented in Table 5.14. As shown in Table 5.14, when we audited the CLIP model released by OpenAI, we obtained a 0% TDR, which indicates that the pretrained CLIP model was not trained on our published data. If it is true that the CLIP model is not, this result empirically confirms the upper bound on the false-detection rate of our method. When we fine-tuned the CLIP model by the marked Flickr30k dataset, acc increased from 80.73% to 88.44% while TDR increased to 100%, which demonstrates that our method is highly effective to detect the use of published data in the fine-tuned CLIP even when it is fine-tuned by only 1 epoch. When we fine-tuned the model for more epochs (e.g, 3 epochs), acc did not significantly increase. With more fine-tuning epochs, we still got a TDR of 100% but a smaller RCD. Fine-tuning by more epochs made the model memorize the fine-tuning samples more and thus we needed fewer queries to the model in the detection step.

5.4 Instance-level Data-Use Auditing

In this section, we consider the more challenging auditing scenarios in which the data owner audits a small amount of data instances (i.e., q is small or even $q = 1$).

5.4.1 Auditing Image Classifiers

5.4.1.1 Experimental Setup

Datasets: We used three visual benchmark datasets, namely CIFAR-100 [72], TinyImageNet [75], and ImageNet [34].

- CIFAR-100: CIFAR-100 is a dataset containing 60,000 images of $3 \times 32 \times 32$ dimensions partitioned into 100 classes. In CIFAR-100, there are 50,000 training samples and 10,000 test samples.
- TinyImageNet: TinyImageNet is a dataset containing images of $3 \times 64 \times 64$ dimensions

partitioned into 200 classes. In TinyImageNet, there are 100,000 training samples and 10,000 validation samples that we used as test samples.

- ImageNet: ImageNet is a large-scale dataset containing images partitioned into 1,000 classes. There are 1,281,167 training samples and 50,000 validation samples that we used as test samples.

Unlike the dataset-level auditing experiments in Sec. 5.3.1.1, we did not include the relatively simple benchmark dataset CIFAR-10. Instead, we additionally considered ImageNet to study the performance of our approach on large-scale datasets.

Data-marking setting: In each experiment, we simulated how the data owner would mark her data and how the ML practitioner would assemble his training dataset by preparing a marked labeled training dataset, as follows: To prepare a marked labeled training dataset, we first uniformly at random sampled 500 training samples for CIFAR-100, 1,000 for TinyImageNet, or 1,000 for ImageNet, as the data instances that we audited. However, for a fixed value of q, each subset of q audited samples were treated independently (as if from a different data owner; i.e., we applied our test for each subset of size q separately). Unlike the dataset-level auditing experiments where q was fixed at 10% of the training samples, we evaluated instance-level auditing with $q \in \{1, 2, 4, 8, 16, 32, 64\}$ to study detection performance under varying q. Taking each subset of q audited samples as X, we applied our data-marking algorithm to generate its published version X' and the hidden information $\overline{X'}$. By default, we set $n = 1000$. When applying the data-marking algorithm on each data instance from X, we set $\epsilon = 10$ for CIFAR-100 or TinyImageNet, or $\epsilon = 25$ for ImageNet when the pixel range of an image is $[0, 255]$. Following the experimental setup in Sec. 5.3.1.1, in Eq. (5.4), we defined utility distance function by $\Delta_{\mathsf{uti}}(x_i^j, x_i) = \left\| x_i^j - x_i \right\|_\infty$ and the distance function by $\Delta_{\mathsf{dis}}(x_i^j, x_i^{j'}) = \left\| \mathsf{Enc}(x_i^j) - \mathsf{Enc}(x_i^{j'}) \right\|_2$, where we used ResNet-18 pretrained on ImageNet as Enc, as our default. We applied a two-step method to approximately solve Eq. (5.4). First, we generated n unit vectors of the same dimension as the output dimension of the feature extractor Enc (e.g., 512 dimensions for a pretrained ResNet-18) such that the

minimum pairwise distance among the n unit vectors was maximized. Second, we crafted the j-th mark such that the dot product between the feature vector of the j-th marked version (prepared by Enc) and the j-th unit vector was maximized while satisfying the constraint (i.e., Eq. (5.4b)). As such, we prepared a marked, labeled, training dataset used to train a classifier, by replacing each audited data instance in the dataset with its published version and assigning it a correct label (i.e., using its original label).

Model training setting: In each experiment, we simulated how the ML practitioner would develop an image classifier by training it from scratch on a training dataset prepared as described. We trained the classifier on the marked dataset, following the training setting described in Sec. 5.4.1.1, except that the model was trained by 100 epochs to achieve better accuracy. The impact of the number of epochs on model accuracy and auditing performance has been studied in Sec. 5.3.1.4. We used ResNet-18 [54] as the default model architecture for CIFAR-100 and TinyImageNet, and used a larger model, ResNet-50, as the default for ImageNet.

We report the accuracies (acc) of the classifiers trained on the marked datasets (i.e., the fraction of test samples correctly predicted) and differences between the accuracies of classifiers trained on marked datasets and those of classifiers trained on clean datasets, in Table 5.15. The classifiers trained on the marked datasets preserved good utility, i.e., their acc values were similar to those trained on clean datasets, which demonstrated that the marked data instances preserved good utility of their original versions.

Data-use detection setting: In each experiment, we simulated how the ML practitioner would deploy his classifier and how a data owner would detect the use of her marked image instance, as follows: Unless otherwise specified, we assumed that the model returned a vector of confidence scores (i.e., a multi-dimensional vector where each component represents the predicted probability for the associated class, also described as CG in Sec. 5.3.1.1) in response to an input image, though we also considered settings where the output was a label (i.e., $\bar{\text{C}}$G described in Sec. 5.3.1.1) or a label with its associated confidence score (i.e., CG under Top-1). Using the marked image set X' and the associated hidden set $\overline{X'}$ generated from the data-

marking setting, the data owner applied our data-use detection algorithm to test if an image classifier was trained using X'. When applying the data-use detection algorithm, we used the black-box membership inference method defined in Sec. 5.3.1.1. In the implementation of membership inference applied in our data-use detection, we set $\pi = 16$ for CIFAR-100 and TinyImageNet, and $\pi = 64$ for ImageNet, as the default. We set $\alpha' = 0.001$, and considered $\alpha = 0.05$, $\alpha = 0.01$, and $\alpha = 0.002$ in the data-use detection algorithm. Recall that α is the bound on FDR of the data auditing method. For example, setting $\alpha = 0.002$ guarantees that $\mathsf{FDR} \leq 0.2\%$.

Baselines: To our knowledge, there is no existing work on instance-level, proactive, data-use auditing of ML. Black-box membership inference can be used to *passively* infer data-use in an ML model but it does not provide a bounded FDR. In addition, membership inference assumes the availability of auxiliary data from the same distribution as the training samples and/or at least one reference model that is trained on a dataset similar to the training set of the tested model. We consider the state-of-the-art black-box membership inference methods, namely Attack-P [171], Attack-R [171], LiRA [15], and RMIA [178] as our baselines. We summarize the limitations of these membership inference methods applied to data-use auditing in Table 5.17. We provide detailed descriptions of the baselines and their implementations as follows.

Given a data instance x and its associated label y, an ML model f (i.e., a classifier), and access to a set of labeled auxiliary data A and some reference models F', these membership inference attacks compute a score $\mathsf{MIA}_{A,F'}(x, y, f)$ and then compare it to a threshold ψ. Membership inference attacks infer that x is used in training f if $\mathsf{MIA}_{A,F'}(x, y, f) \geq \psi$. Here ψ controls the empirical FDR of a membership inference attack method. The computation of $\mathsf{MIA}_{A,F'}(x, y, f)$ by these membership inference attacks is presented in Table 5.16 [178, Table 1].

For each experiment involving a comparison with baselines, we randomly divided a dataset's training samples into two equal halves. One half (e.g., 25,000 CIFAR-100 training samples or 50,000 TinyImageNet training samples) was used to train the classifier that we

audited, while the other half was used to train reference models required for membership inference. The test samples of each dataset were randomly split into two equal halves: One half was designated as auxiliary data, while the other half was used to empirically measure FDR of a black-box membership inference method. For membership inference methods that incorporate data augmentation (e.g., LiRA and RMIA), we set the number of augmentations to 2 in these methods, following their default settings [15, 178]. To ensure a fair comparison, we also set $\pi = 2$ for our method when benchmarking against these baselines, and limited the baselines' queries to the audited ML model to be $\pi \times n$ (for $q = 1$). Since our data-use detection method allows a data owner to stop querying the audited ML model early, our method poses $\leq \pi \times n \times q$ queries to each model. Therefore, under these settings, our method had an equal or lower query cost than the baselines.

We implemented these membership inference attacks in PyTorch [109] based on their respective papers and available open-source codes.[9][10] We considered a setting where at most one reference model was accessible, i.e., $|F'| = 1$. To simulate the access to an auxiliary dataset A and a reference model F' ($|F'| = 1$), we randomly split the training set of a dataset into two non-overlapping halves, e.g., each half included 25,000 CIFAR-100 training samples or 50,000 TinyImageNet training samples. The first half was used to train an audited ML model f and the second half was used to train a reference model f'. We also randomly split the test set of a dataset into two non-overlapping halves. The first half was used as A and the second half was used as "a non-member set" to empirically measure FDR. In Table 5.16, the original versions of these methods replace $\left[f(x)\right]_y$, $\left[f'(x)\right]_y$, $\left[f(\acute{x})\right]_{\acute{y}}$, and $\left[f'(\acute{x})\right]_{\acute{y}}$ by a metric (e.g., SM-Taylor-Softmax [33]) based on model logits (i.e., the outputs before a Softmax layer). However, in our data-use auditing setting, the model outputs are vectors of confidence scores (i.e., the outputs after the Softmax layer), as described in Sec. 5.4.1.1. Therefore, such a metric is not applicable in a data-use auditing setting. We searched for an threshold ψ to ensure that the empirical FDR of a membership inference attack was merely

[9] https://github.com/tensorflow/privacy/tree/master/research/mi_lira_2021

[10] https://github.com/privacytrustlab/ml_privacy_meter/

below a specified level (e.g., $\mathsf{FDR} \leq 1\%$). Using the identified ψ, we then calculated TDR of the membership inference method for the specific level of FDR.

Metric: We used the following metrics to evaluate the methods:

- Test accuracy (acc): Please see its description in Sec. 5.3.1.1.
- True-detection rate (TDR): Please see its description in Sec. 5.3.1.1.
- Relative cost for detection (RCD): Please see its description in Sec. 5.3.1.1.

5.4.1.2 Experimental Results

Main results: Our results of auditing data-use in image classifiers are shown in Fig. 5.8. Under the default setting, our TDR ($q = 1$) for auditing CIFAR-100 (TinyImageNet) instances was 28.21% (18.09%), 11.59% (6.02%), and 3.11% (1.39%), when the bounds on FDR were set as 5%, 1%, and 0.2%, respectively. When the output of a classifier included both the predicted label and its associated confidence score, the TDR remained comparable to that under the default setting. When only the predicted label was available, our TDR ($q = 1$) was only marginally above the FDR level. However, when the data owner possessed more data instances (i.e., $q > 1$) and all of them were used in training, our method became significantly more effective, even when the outputs of the audited classifiers were labels only ("Label"). For ImageNet, our TDR ($q = 1$) was only slightly higher the FDR level at $q = 1$, but clearly improved on it by $q = 16$.

Fig. 5.9 presents the results of scenarios where a data owner had $q = 10$ data instances and $q' \in \{1, \ldots, q\}$ were used in training. As in Fig. 5.9, increasing q' quickly improved TDR, noticeably improving over the FDR bound for both confidence vectors ("All") and classification confidences ("Highest"), even when $q' = 2$ or 3.

We plot the cumulative distribution function (CDF) of G in the case $q = 1$ and $n = 1000$, under the condition that data use is detected, in Fig. 5.10. In Fig. 5.10, the area under the curve (AUC) represents the average query cost saved when the audited data instance was detected. For CIFAR-100, the AUCs were 510.24 and 107.78 when $\mathsf{FDR} \leq 5\%$ and $\mathsf{FDR} \leq 1\%$ respectively. For TinyImageNet, the AUCs were 472.42 and 96.81 when $\mathsf{FDR} \leq 5\%$ and

$\mathsf{FDR} \leq 1\%$ respectively. When we set $\alpha = 0.002$, it needed to query all the marked data instances (i.e., $G = 1000$) in order detection data use and so we did not have cost savings in this setting.

Empirical false-detection: We empirically evaluate FDR of our method by training image classifiers on datasets excluding the audited data samples. The results on our empirical FDR are shown in Table 5.18. As shown in Table 5.18, the empirical FDR were less than the bound α on false-detection. These results empirically confirm the upper bounds on FDR of our method.

Comparison with baselines: We compare our method with the state-of-the-art membership inference methods, namely Attack-P [171], Attack-R [171], LiRA [15], and RMIA [178] when $q = 1$. Attack-R, LiRA, and RMIA assume the ability to train at least one ML model, known as a *reference model*, from these samples. In those works that proposed Attack-R, LiRA, and RMIA, such reference models are assumed similar to the audited model (i.e., they are trained on a dataset similar to the training dataset of the audited model).

In our implementation of Attack-R, LiRA, and RMIA, we considered a more realistic setting where only one reference model was used in membership inference. We also considered settings where the reference model is not similar enough to the audited model, by constructing a dataset used to train the reference model differently from the training dataset of the audited model. We constructed such a "different" dataset by decreasing its size $\acute{m}$ and/or introducing class imbalance (i.e., the number of data samples per class was unequal). Specifically, the class proportions were drawn from a Beta distribution [68] where we set its two parameters as the same value β that controls the degree of imbalance. A larger value β leads to greater skewness in the class proportions.

The comparison results are presented in Fig. 5.11 and Fig. 5.12. Fig. 5.11 shows the auditing/inference results on CIFAR-100: When Attack-R, LiRA, and RMIA used a reference model similar to the audited model, our method achieved a TDR comparable to those of Attack-R, LiRA, and RMIA under the same FDR level. However, when the reference model was not similar enough to the audited one (e.g., by decreasing $\acute{m}/m$, where m is

the size of the training set of the audited model, and/or increasing β), the performance of these membership inference methods significantly degraded, which was also confirmed by their works [15, 171, 178]. For example, the state-of-the-art membership inference method, namely RMIA, had TDR of 15.27%, 2.25%, and 0% under FDR $\leq$ 5%, FDR $\leq$ 1%, and FDR $\leq$ 0.2%, respectively, when we set $\acute{m}/m = \frac{1}{8}$ and $\beta = 4$, much lower than ours.

The performance of these three membership inference methods were highly affected by the reference models. Of course, as shown in the previous works (e.g., [15, 171, 178]), when more reference models can be trained and used in membership inference, these methods would achieve a better inference result (i.e., a higher TDR). However, in a realistic scenario of data-use auditing, it is costly to train a reference model and challenging to collect a dataset used to train the reference model that is similar to the training dataset of the audited model. Attack-P does not require a reference model but its TDR was much lower than ours under the same FDR. We have similar observation and conclusion from the results on TinyImageNet, as shown in Fig. 5.12. More importantly, all these membership inference methods do not provide a bound on the FDR. This limits the application of membership inference methods in auditing data-use of ML models.

Robustness against countermeasures: We study the effectiveness of our data-use auditing method when the ML practitioner applies countermeasures in the training pipeline to defeat our method. We considered those countermeasures introduced in Sec. 5.3.1.4: (1) adding Gaussian noise into the training samples (Rand(σ)); (2) using our marking algorithm (with the default hyperparameters) to perturb training samples (MarkPerturb); and (3) differentially private stochastic gradient descent (DPSGD). We did not include the other countermeasures because they either cannot mitigate our approach or significantly degrade the model's utility. We additionally considered the following countermeasures:

- *Image denoising*: The ML practitioner applies image denoising techniques [53, 60] on the training samples in order to remove their added marks. We consider three commonly used denoising methods: (1) Gaussian smoothing, (2) median smoothing, and (3) general smoothing.

- *Regularization*: The ML practitioner applies regularization techniques, e.g., by adding a penalty proportional to the squared values of the model parameters, to reduce memorization of training samples. He controls the regularization strength by tuning the weight decay (denoted as ω) parameter in the SGD optimizer.

We report our robustness results in Fig. 5.13, Table 5.19, Fig. 5.14, and Fig. 5.15. As shown in Fig. 5.13 and Table 5.19, both perturbation methods decreased our TDR: "Remarking" decreased TDR ($q = 1$) from 28.21% to 12.60% under FDR $\leq$ 5% but the accuracy acc on average dropped by 3.90 percentage points. Adding Gaussian noise with a larger σ made our TDR ($q = 1$) closer to the FDR level but it also sacrificed more model accuracy. For example, adding Gaussian noise with $\sigma = 25$ decreased TDR ($q = 1$) to a level close to FDR but at a cost of 10.11 percentage points to acc. As shown in Table 5.19, image smoothing decreased our TDR but significantly decreased the utility of the trained models. For example, general smoothing reduced TDR ($q = 1$) to 14.49% under FDR $\leq$ 5% but acc dropped by around 11 percentage points. As shown in Fig. 5.14 , when applying DPSGD, when we set a larger σ, the trained classifier memorized its training samples less and thus our TDR ($q = 1$) decreased under the same level of FDR. For example, under FDR $\leq$ 5%, our TDR ($q = 1$) decreased from 28.21% to 9.21% when we increased σ from 0 to 2×10^{-3} ($\sigma = 0$ corresponds to the non-private setting). However, acc of the trained classifiers decreased from 75.53% to 65.82% as σ grew. We have similar observations from the auditing results (see Fig. 5.15) when applying stronger regularization techniques in model training. To summarize, none of these countermeasures defeated our auditing method without significantly sacrificing the utility of the trained ML model.

When the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became more effective, i.e., our TDR increased with q, even when the ML practitioner applied a strong countermeasure. For example, when DPSGD with $\sigma = 2 \times 10^{-3}$ was used to train the ML model, our TDR achieved 24.34% ($q = 8$) and 85.34% ($q = 64$) under FDR $\leq$ 5% (compared with 9.21% for $q = 1$).

Across different model architectures: We study the effectiveness of our data-use audit-

ing method on auditing data use ($q = 1$) in image classifiers of different model architectures, namely ResNet-18, ResNet-34 [54], WideResNet-28-2 [177], VGG-16 [130], and ConvNetBN [65]. We only considered CIFAR-100 and trained these classifiers using the same training algorithm (please see Sec. 5.4.1.1). Our auditing results are shown in Table 5.20. Our TDR ranged from 20.80% to 29.90%, from 7.20% to 11.89%, and from 1.53% to 3.25% when the bounds on FDR were set as 5%, 1%, and 0.2%, respectively.

The impact of the utility-preservation parameter ϵ: We study the impact of the utility-preservation parameter ϵ on the performance of our data auditing method ($q = 1$), by changing ϵ from 6 to 20. ϵ controls the visual quality (utility) of the marked image. We present examples of marked CIFAR-100 images under different ϵ in Fig. 5.6. Our auditing results for varying ϵ are presented in Fig. 5.16. As in Fig. 5.16, a larger ϵ led to a higher TDR under the same FDR bound, which shows a trade-off between utility-preservation and TDR.

Impact of using a different n: We consider a setting where our marking algorithm generated $n = 200$, $n = 500$, or $n = 5{,}000$ marked data per raw CIFAR-100 data instance. Our results ($q = 1$) are presented in Table 5.21. By comparing with the results of $n = 1{,}000$, we have the following observations: When we set an FDR bound much larger than $\frac{1}{n}$, e.g., FDR $\leq 5\%$ or FDR $\leq 1\%$, TDR did not change much when increasing n. However, when we considered FDR $\leq 0.2\%$, TDR increased from 3.11% to 4.43% if we increased n from 1,000 to 5,000. In addition, setting a larger n allowed the data owner to detect her data-use under a lower FDR bound (e.g., FDR $\leq 0.1\%$). But setting a larger n also brought a higher cost in marked data generation and data-use detection, as shown in Fig. 5.17 and Fig. 5.18.

Impact of using data augmentation in data-use detection: We plot TDR ($q = 1$) of our auditing method using varying number π of data augmentations in Fig. 5.19. When we used a larger number of data augmentations in the data-use detection algorithm, our method achieved a higher TDR under the same FDR bound. However, a larger number of data augmentations requires more (black-box) queries to the ML model, and thus it brings a

higher query cost.

Factor impacting auditability: We study a factor that might impact the auditability of an image. Given an ML model f trained using a (marked) image x' ($q = 1$) and membership inference method $\mathsf{MI}^f(\cdot)$, we measured the auditability of an image instance x' by its $\mathsf{rank}\big(x', \overline{X'}, \mathsf{MI}^f\big)$, i.e., a higher $\mathsf{rank}\big(x', \overline{X'}, \mathsf{MI}^f\big)$ indicates more auditability. We considered the difference between the loss of a model that is not trained on the audited data instance x' and the loss of the model f that is trained on x', denoted as $\ell(f_{-x'}, x') - \ell(f, x')$ where ℓ denotes the loss function and $f_{-x'}$ denotes the model that is not trained on x'. The loss difference indicates how a marked data instance is vulnerable to a membership inference attack [15], i.e., a data instance with larger loss difference is more vulnerable to membership inference. We plot the distribution of $(\mathsf{rank}\big(x', \overline{X'}, \mathsf{MI}^f\big), \ell(f_{-x'}, x') - \ell(f, x'))$ in Fig. 5.20. We calculated the Pearson correlation coefficients [30] between $\mathsf{rank}\big(x', \overline{X'}, \mathsf{MI}^f\big)$ and $\ell(f_{-x'}, x') - \ell(f, x')$. The resulted coefficients for CIFAR-100 and TinyImageNet are 0.331 and 0.295 respectively. These results indicate a weak positive linear correlation between $\mathsf{rank}\big(x', \overline{X'}, \mathsf{MI}^f\big)$ and $\ell(f_{-x'}, x') - \ell(f, x')$. In other words, it would be easier to audit the use of a data instance with a larger loss difference. This observation is consistent with that from previous works on privacy attacks [15]. We leave exploring other factors impacting auditability of a data instance as a direction for future work.

Ablation study on data-marking algorithm: We study the impact of two components of our data-marking algorithm: (1) optimizing the n unit vectors such that their minimum pairwise distance is maximized (compared to generating them by sampling randomly), and (2) optimizing the added marks such that the distance between any pair of generated marked data is maximized (compared to generating added marks randomly). We denote the method of generating the added marks randomly (i.e., each pixel of a mark is uniformly at random sampled from $\{-\epsilon, \epsilon\}$) as `RM`. We denote the method of generating marks by generating random unit vectors and optimizing the marks to maximize the distance between any pair of marked data, as `RUV+OM` (only component (2) included). We denote the method of generating

marks by both optimizing the unit vectors and the marks, as OUV+OM (both component (1) and (2) included). The results ($q = 1$) are presented in Table 5.22. As in Table 5.22, OUV+OM achieved the highest TDR under the same level of FDR. Compared with RM, OUV+OM improved by 5.44 percentage points, 3.36 percentage points, and 1.01 percentage points, under the false-detection bounds of 5%, 1%, and 0.2%, respectively. Such improvement demonstrates that a useful feature extractor helps in generating "effective" marked data. This is because using a feature extractor can embed marks into the high level features of an image such that the high level features of n marked data are maximally different and so it is easier for a membership inference method to distinguish a model trained on one but not the others. Compared with RUV+OM, OUV+OM achieved a slightly higher TDR. Random sampling can generate unit vectors whose minimum pairwise distance is large enough and thus optimizing unit vectors only made marginal improvement.

Ablation study on data-use detection algorithm: We study the effectiveness of using the rank of the added mark of the published data in detecting data use. In other words, we measured the "memorization" scores of the added marks from n generated marked data, then estimated the rank of the added mark of x', and detected data use based on the estimated rank. This is the adaption of the idea from Carlini et al.'s work [16] to our setting. Our results ($q = 1$) in Table 5.23 show that the rank of the added mark did not provide strong evidence of data-use in visual models, i.e., TDR values were low and close to the level of FDR. This is because the visual model memorizes the whole marked data instance in its training rather than the added mark.

5.4.2 Auditing Visual Encoders

5.4.2.1 Experimental Setup

Datasets: We used CIFAR-100 [72] and TinyImageNet [75].

Data-marking setting: We followed the default data-marking setup described in Sec. 5.4.1.1 to prepare marked training datasets, without labels needed.

Model training setting: We trained the visual encoder by SimCLR algorithm [23], which

we have described in Sec. 5.3.2.1. We used a larger deep neural network ResNet-34 as the architecture of the visual encoder.

Data-use detection setting: We assumed that the data owner can obtain a vector of features by providing her marked image or its augmented version as input to the visual encoder. Using the marked images X' and the associated hidden set $\overline{X'}$ generated from the data-marking setting, we applied our data-use detection algorithm to test if a visual encoder was trained using X'. When applying the data-use detection algorithm, we followed the previous works (e.g., [62, 83, 186]) to define the black-box membership inference method, which has been introduced in Sec. 5.3.2.1. In the implementation of membership inference applied in our data-use detection, we set $\pi = 64$ as the default. We considered $\alpha = 0.05$, $\alpha = 0.01$, and $\alpha = 0.002$ in the data-use detection algorithm.

Metric: We used TDR to measure the effectiveness of a data-use auditing method. Unlike experiments in Sec. 5.3.2, we primarily study the the auditing performance of our approach under varying small values of q in instance-level auditing scenarios. The impact of epochs on model utility and auditing performance has been studied in Sec. 5.3.2.

5.4.2.2 Experimental Results

Our results on auditing visual encoders are presented in Fig. 5.21. When $q = 1$, our TDR ranged from 10.51% to 14.02%, from 2.40% to 3.78%, and from 0.34% to 0.56% when the bounds on FDR were set as 5%, 1%, and 0.2%, respectively. Although our TDR on auditing visual encoders were significantly better than those from a "random guessing" method, they were lower than those on auditing image classifiers (see Fig. 5.8). Because visual encoders are trained to learn the general representations of images and thus they memorizes their training samples less. When the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became significantly more effective, i.e., our TDR increased with q.

5.4.3 Auditing CLIP and BLIP

In addition to studying CLIP in dataset-level auditing scenarios in Sec. 5.3.4, we consider BLIP, another vision-language model, to evaluate the performance of our approach in instance-level auditing scenarios.

5.4.3.1 Experimental Setup

Datasets: We used the Flickr30k dataset.

Data-marking setting: We first randomly sampled 2,500 images with textual descriptions as training samples and others as test samples. From 2,500 training samples, we randomly selected 250 samples as the audited samples. However, for a fixed value of q, each subset of q audited samples were treated independently (as if from a different data owner; i.e., we applied our test for each subset of size q separately). We applied our data-marking algorithm to generate its published version X' and the associated hidden information set $\overline{X'}$, where we set $\epsilon = 10$, $n = 1000$, and used ResNet-18 pretrained on ImageNet as Enc. As such, we prepared a marked training dataset by replacing each audited image with its published version and assigning it to its original textual description.

Model training setting: We fine-tuned the pretrained CLIP (`ViT-B/32`) released from OpenAI on marked datasets, following the same setting in Sec. 5.3.4.1. We fine-tuned the pretrained BLIP image captioning base model on marked datasets prepared from the data-marking setting. We applied the AdamW optimizer [88] with a learning rate of 5×10^{-5} to fine-tune the BLIP model on a mini-batch of 8 training samples at each iteration.

Data-use detection setting: For CLIP model, we assumed that the data owner could obtain feature vectors by providing her marked image and its corresponding textual description as inputs to the CLIP model. Using the marked images X' and the associated hidden set $\overline{X'}$ generated from the data marking, she applied our data-use detection algorithm to test if a CLIP was trained/fine-tuned using X'. When applying the data-use detection algorithm, we followed the previous works (e.g., [71]) to define the black-box membership inference method, which has been introduced in Sec. 5.3.4.1: Given an input image and its textual

description, we obtained their corresponding feature vectors (one for image and the other for its textual description) by CLIP; Then we used the cosine similarity between the two feature vectors as the "memorization" score of the input image.

For BLIP model, we assumed that the data owner could obtain the probability distributions of the output by providing her marked image. Using the marked images X' and the associated hidden set $\overline{X'}$ generated from the data marking, she applied our data-use detection algorithm to test if a BLIP image captioning model was trained/fine-tuned using X'.

We considered $\alpha = 0.05$, $\alpha = 0.01$, and $\alpha = 0.002$ in the data-use detection algorithm.

Metric: We used TDR to measure the effectiveness of a data-use auditing method.

5.4.3.2 Experimental Results

Our results on auditing the data-use in CLIP, fine-tuned by only one epoch, are shown in Fig. 5.22. With $q = 1$, we achieved TDR of 9.58%, 2.82%, and 0.48% under FDR $\leq$ 5%, FDR $\leq$ 1%, and FDR $\leq$ 0.2%, respectively, which were significantly better than random guessing. When the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became much more effective, i.e., our TDR increased with q. For example, we achieved TDR of 67.35%, 39.78%, and 16.45% under FDR $\leq$ 5%, FDR $\leq$ 1%, and FDR $\leq$ 0.2%, respectively, when the data owner had $q = 64$ data instances.

We also report our auditing results when the CLIP model was fine-tuned on the marked datasets for more than one epoch, in Fig. 5.23. With $q = 1$, when we fine-tuned the CLIP model by more epochs, its acc slightly increased from 85.44% to 86.57% and TDR also increased, e.g., from 9.60% to 16.38% under FDR $\leq$ 5%. Fine-tuning CLIP by more epochs makes it memorize its training samples more and thus it is easier to audit data-use (i.e., a higher TDR), which was also observed by previous works on dataset-level data-use auditing (e.g., [62]). When the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became significantly more effective, i.e., our TDR increased with q.

Our results on auditing the data-use in BLIP, fine-tuned by only one epoch, are also included in Fig. 5.22. With $q = 1$, we achieved TDR of 13.07%, 3.43%, and 0.55% under FDR $\leq$ 5%, FDR $\leq$ 1%, and FDR $\leq$ 0.2%, respectively. When the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became significantly more effective, i.e., our TDR increased with q. For example, we achieved TDR of 94.72%, 82.00%, and 57.15% under FDR $\leq$ 5%, FDR $\leq$ 1%, and FDR $\leq$ 0.2%, respectively, when the data owner had $q = 64$ data instances.

We also report our auditing results when the BLIP models were fine-tuned on the marked datasets for more than one epochs, in Fig. 5.24. With $q = 1$, when we fine-tuned the BLIP model by more epochs, its BLEU score decreased due to overfitting, but TDR increased, e.g., from 13.07% to 24.42% under FDR $\leq$ 5%. Fine-tuning BLIP by more epochs makes it memorize its training samples more and thus it is easier to audit data-use (i.e., a higher TDR). Again, when the data owner had more data instances (i.e., $q > 1$) and all of them were used in training, our method became significantly more effective, i.e., our TDR increased with q.

5.5 Discussion

5.5.1 Auditing Text Data in Instance-Level Auditing Scenarios

Our work mainly focuses on image instance in instance-level auditing scenarios. While our method could be applied into text data, such application presents several challenges. A key difficulty lies in generating n distinct marked instances for each raw data sample when n is large (e.g., $n = 1{,}000$). In our dataset-level auditing scenarios where we set $n = 2$, we applied a paraphraser model to paraphrase a text data to generate two marked versions. However, when $n = 1{,}000$, it is challenging to paraphrase a raw text data for n times to generate n distinct marked text data while ensuring that all of them preserve the semantic meaning.

5.5.2 Minimal Number of Marked Data Required in Auditing

The minimal number of marked (published) data for which our method can detect its use depends on two factors: the memorization of training data by the ML model and the effectiveness of (contrastive) membership inference. For example, as shown in Sec. 5.3.1.2 and Sec. 5.4.1.2, the CIFAR-100 and TinyImagenet classifiers memorized their training samples more than the CIFAR-10 classifier, and so the data owner needed much less marked data to audit for data use in the CIFAR-100 and TinyImagenet classifiers than in the CIFAR-10 classifier. The effectiveness of (contrastive) membership inference also affects the minimal number of data items for which our method can detect use, i.e., a stronger membership inference method will allow our method to detect the use of fewer data. Therefore, we believe that any developed stronger membership inference methods in the future will benefit our technique.

5.5.3 Adaptive Attacks to Data Auditing Applied in Foundation Models

Once the ML practitioner realizes that the data auditing is being applied, he might utilize adaptive attacks aiming to defeat the auditing method when training his foundation models. Some adaptive attacks we considered for the image classifier (see Sec. 5.3.1.4) and Sec. 5.4.1.2 like early stopping, regularization, and differential privacy, can be used to mitigate the memorization of training/fine-tuning samples of foundation models. Therefore, these adaptive attacks could degrade the effectiveness or efficiency of our detection method. In addition, there are some methods used to mitigate membership inference in LLMs, e.g., model parameter quantization/rounding [107]. Any defense against membership inference in foundation models can be used as an adaptive attack. However, the application of these adaptive attacks will decrease the utility of the foundation models [107].

Since developers of foundation models usually aim to develop a powerful foundation model, they might hesitate to apply these adaptive methods since they will lose some model utility. As such, our data auditing method can pressure those developers of large foundation models to seek data-use authorization from the data owners before using their data.

5.5.4 Cost of Experiments on Foundation Models

In our experiments on auditing data use in foundation models (e.g., Llama 2 in Sec. 5.3.3 and CLIP in Sec. 5.3.4 and Sec. 5.4.3), we only considered model fine-tuning due to our limited computing resources. From the results shown in Sec. 5.3.3.2, Sec. 5.3.4.2, and Sec. 5.4.3.2, our proposed method achieves good performance on detecting the use of data in fine-tuning Llama 2 and CLIP. We do believe that the effective detection performance of our method can be generalized to other types of foundation models and the settings where we audit the use of data in pretrained foundation models. This is because large foundation models memorize their training samples and thus are vulnerable to membership inference and other privacy attacks, as shown by existing works (e.g., [17, 71, 83, 107, 126]).

5.5.5 Toward Verifiable Machine Unlearning

One direct application of our data-auditing method is to verify machine unlearning. Machine unlearning is a class of methods that enable an ML model to forget some of its training samples upon the request of their owners. While there are recent efforts to develop machine unlearning algorithms [11], few focus on the verification of machine unlearning, i.e., verifying if the requested data has indeed been forgotten by the target model [133]. Our proposed method can be a good fit for verifying machine unlearning. Specifically, each data owner utilizes our marking algorithm to generate published data and hidden data. Upon the approval of data owners, a ML practitioner collects their published data and trains an ML model that can be verified by the data owner using our detection algorithm. If a data owner sends a request to the ML practitioner to delete her data from the ML model, the ML practitioner will utilize a machine unlearning algorithm to remove her data from his ML model and then inform the data owner of the successful removal. The data owner can utilize the detection algorithm to verify if the updated ML model still uses her published data. Our results in Sec. 5.3.1.2 show that our auditing method remains highly effective even when multiple data owners audit their data independently.

5.5.6 Proving a Claim of Data Use

Though our technique enables a data owner to determine whether an ML practitioner used her data without authorization, it alone does not suffice to enable the data owner to convince a third party.

To convince a third party, the data owner should commit, prior to data publication, to the raw data, all generated marked data (including both the published data and those kept secret), and the seeds to a random number generator for the selection of the published data. For example, she can escrow a cryptographic commitment to these data with the third party. One such seed to a random number generator could be set to be a cryptographic hash of the raw data. This commitment ensures that the data owner has followed the prescribed data-marking procedure and introduced the randomness required for the false-detection guarantees.

Upon detecting potential use of her data by an ML practitioner, the data owner must additionally demonstrate that the publication time of the released data precedes the deployment time of the ML model, ensuring that the model could have incorporated the published data during training. Otherwise, the trained model could not have used the data in its training. Then the data owner can open these commitments to enable the third party to perform our hypothesis test on the ML model itself, for example. To enable a third party to replicate the data-owner's test result exactly, the data owner could provide the seed to a random number generator to drive the sequence of selections (WoR) from $\overline{X'}$ in the test (see Sec. 5.2.2). However, to protect an ML practitioner from being framed by a malicious data owner, the data owner should be unable to freely choose this seed; e.g., it could be set to be a cryptographic hash of the commitments to the data.

5.6 Chapter Summary

In this chapter, we proposed a general framework that enables a data owner to audit whether her data has been used in training a model, at both the dataset and instance levels. Our auditing method builds upon arbitrary membership-inference techniques and integrates

them into an anytime-valid statistical test that we design, which enables the data owner to continuously accumulate data-use evidence through querying the model and adaptively stop at any time while maintaining a quantifiable and provably bounded false-detection rate. Through extensive evaluations on diverse ML models—including image classifiers, visual encoders, LLMs, as well as CLIP and BLIP models, we demonstrate the effectiveness and generality of our approach across a wide range of ML tasks and settings.

Table 5.2: Superclass of TinyImageNet generated by querying to ChatGPT.

Superclass	Class
Aquatic Animals	European fire salamander, bullfrog, jellyfish, sea slug, spiny lobster, tailed frog, brain coral, goldfish, sea cucumber, American lobster
Land Animals	koala, black widow, trilobite, scorpion, tarantula, centipede, boa constrictor, American alligator, dugong
Birds	black stork, goose, king penguin, albatross
Domestic Animals	Chihuahua, golden retriever, Persian cat, Yorkshire terrier, German shepherd, Egyptian cat, standard poodle, Labrador retriever, tabby
Wild Animals	lion, chimpanzee, lesser panda, orangutan, brown bear, baboon, cougar, African elephant
Insects and Arachnids	dragonfly, monarch, walking stick, grasshopper, ladybug, sulphur butterfly, fly, mantis, cockroach, bee
Clothing and Accessories	academic gown, bikini, sandal, poncho, military uniform, cardigan, fur coat, miniskirt, swimming trunks, bow tie, kimono, vestment, sombrero, apron
Transportation	trolleybus, gondola, lifeboat, jinrikisha, bullet train, convertible, school bus, police van, sports car, beach wagon, limousine, moving van
Buildings and Structures	triumphal arch, cliff dwelling, butcher shop, fountain, steel arch bridge, barbershop, suspension bridge, barn, freight car, water tower, viaduct, dam, obelisk, beacon
Household Items	beaker, snorkel, candle, Christmas stocking, dumbbell, turnstile, lawn mower, computer keyboard, parking meter, backpack, scoreboard, water jug, wok, dining table, pay-phone, sewing machine, hourglass, tractor, banister, pole, plate, sock, bathtub, torch, magnetic compass, spider web, frying pan, plunger, drumstick, birdhouse, gasmask, umbrella, stopwatch, rocking chair, teapot, sunglasses, flagpole, teddy, punching bag, beer bottle, lampshade, reel, refrigerator, rugby ball, pill bottle, broom, binoculars, space heater, chest, volleyball, iPod, bucket, maypole, desk, wooden spoon, syringe, remote control
Food	pretzel, ice cream, cauliflower, ice lolly, meat loaf, espresso, potpie, mushroom, guacamole, bell pepper, pizza, orange, pomegranate, mashed potato, banana, lemon
Natural Locations	coral reef, seashore, lakeside, alp, cliff
Miscellaneous Objects	barrel, basketball, potter's wheel, Arabian camel, abacus, neck brace, oboe, projectile, confectionery, bighorn, chain, picket fence, hog, comic book, slug, guinea pig, nail, go-kart, ox, snail, gazelle, organ, altar, crane, pop bottle, bison
Plants and Nature	acorn
Technology and Electronics	cash machine, CD player
Others	brass, thatch, cannon

Table 5.3: Overall performance of our proposed method on different image benchmarks, with an upper bound of $\alpha = 0.05$ on the false-detection rate. All results are averaged over 20 experiments. The numbers in the $\triangle$acc% column are the differences between averaged accuracies of ML models trained on marked datasets and those of ML models trained on clean datasets.

	acc%	$\triangle$acc%	CG		C$\bar{\text{G}}$		$\bar{\text{C}}$G		$\bar{\text{C}}\bar{\text{G}}$	
			TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
CIFAR-10	93.64	-0.05	100%	22.0%	100%	26.7%	100%	42.2%	100%	46.5%
CIFAR-100	74.29	-0.76	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%
TinyImageNet	59.13	-0.16	100%	1.4%	100%	1.3%	100%	5.9%	100%	6.7%

Table 5.4: Comparison between our proposed method and baselines under different rates of $\frac{q'}{m} \in \{1\%, 2\%, 5\%, 10\%\}$. The results of our method come from the setting with least information available to the data owner, i.e., $\bar{\text{C}}\bar{\text{G}}$. In UBW-C, τ is a hyperparameter of its detection algorithm. In the columns of "Labeling known" and "White box", "●" indicates that the information is needed; "○" means that information is not needed; "◐" means that partial information is needed. In the column "bounded FDR", "✔" ("✗") indicates that the method provides (does not provide) a provable bound on the false-detection rate. Results are averaged over 20 experiments. The bold results are the best ones among the compared methods.

		Labeling known	White box	Bounded FDR	1%		2%		5%		10%	
					TDR	acc %	TDR	acc %	TDR	acc %	TDR	acc %
CIFAR-10	Our method ($\bar{\text{C}}\bar{\text{G}}$)	○	○	✔	**40%**	**93.79**	**55%**	**93.71**	**95%**	**93.70**	**100%**	**93.64**
	RData	●	●	✔	5%	93.65	10%	93.56	10%	93.29	20%	93.26
	RData (one mark)	○	●	✔	0%	93.75	0%	93.60	0%	93.42	0%	93.25
	UBW-C ($\tau = 0.25$)	●	○	✗	0%	93.50	0%	93.14	0%	92.67	10%	92.73
	UBW-C ($\tau = 0.20$)	●	○	✗	5%	93.50	40%	93.15	35%	92.46	75%	92.52
CIFAR-100	Our method ($\bar{\text{C}}\bar{\text{G}}$)	○	○	✔	**100%**	**75.01**	**100%**	**74.94**	**100%**	**74.60**	**100%**	**74.29**
	RData	●	●	✔	25%	74.66	70%	74.57	**100%**	73.81	**100%**	73.53
	RData (superclass)	◐	●	✔	20%	74.76	50%	74.46	70%	73.99	95%	73.42
	RData (one mark)	○	●	✔	0%	74.70	0%	74.51	5%	74.05	0%	73.51
	UBW-C ($\tau = 0.25$)	●	○	✗	0%	74.60	0%	74.16	80%	73.30	**100%**	72.32
	UBW-C ($\tau = 0.20$)	●	○	✗	95%	74.60	**100%**	74.33	**100%**	73.21	**100%**	72.47
TinyImageNet	Our method ($\bar{\text{C}}\bar{\text{G}}$)	○	○	✔	**100%**	**59.32**	**100%**	**59.24**	**100%**	**59.17**	**100%**	**59.13**
	RData	●	●	✔	40%	59.14	90%	58.94	**100%**	58.59	**100%**	58.13
	RData (superclass)	◐	●	✔	35%	59.14	70%	59.03	**100%**	58.71	**100%**	58.09
	RData (one mark)	○	●	✔	10%	59.12	5%	58.98	0%	58.61	0%	58.29
	UBW-C ($\tau = 0.25$)	●	○	✗	0%	59.01	0%	58.80	0%	58.43	0%	57.78
	UBW-C ($\tau = 0.20$)	●	○	✗	0%	59.01	0%	58.62	30%	58.41	85%	57.63

Table 5.5: Comparison between our method and RData (which requires knowledge of data labeling), both under an upper bound of $\alpha = 0.05$ on the false-detection rate, when multiple data owners applied data auditing independently. Each owner contributed 10% of the training dataset. Results are the total detections over all detection attempts (by all data owners) in 20 experiments.

	Data owners			
	1	2	5	10
Our method ($\bar{C}\bar{G}$)	100%	100%	100%	100%
RData	100%	95%	64%	45%

Table 5.6: The performance of our method on detecting use of published data in ML model across different architectures, trained on CIFAR-100. All results are averaged over 20 experiments.

	CG		$C\bar{G}$		$\bar{C}G$		$\bar{C}\bar{G}$	
	TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
ResNet18	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%
VGG16	100%	3.0%	100%	3.2%	100%	6.9%	100%	7.1%
ConvNet64	100%	7.2%	100%	29.8%	100%	9.0%	100%	38.9%
ConvNetBN	100%	1.8%	100%	3.2%	100%	3.4%	100%	5.6%
MobileNetV2	100%	3.8%	100%	14.6%	100%	7.4%	100%	21.4%

Table 5.7: The impact of ϵ on the performance of our method applied on CIFAR-100. $\epsilon = 10$ is the default. All results are averaged over 20 experiments.

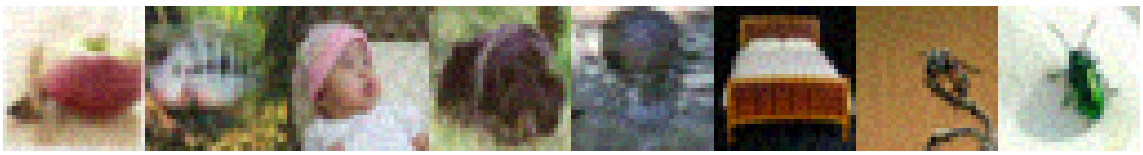
	acc%	CG		$C\bar{G}$		$\bar{C}G$		$\bar{C}\bar{G}$	
		TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
$\epsilon = 2$	74.99	100%	26.9%	100%	28.7%	55%	94.8%	60%	97.7%
$\epsilon = 4$	74.84	100%	8.0%	100%	10.0%	100%	42.5%	100%	43.8%
$\epsilon = 6$	74.64	100%	4.3%	100%	5.0%	100%	15.4%	100%	19.7%
$\epsilon = 8$	74.39	100%	2.5%	100%	2.8%	100%	8.5%	100%	10.8%
$\epsilon = 10$	74.29	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%

(a) Raw images

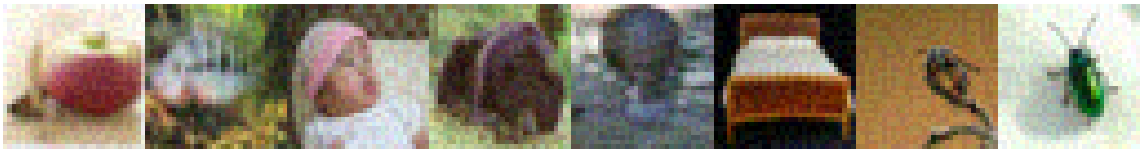
(b) $\epsilon = 2$

(c) $\epsilon = 4$

(d) $\epsilon = 6$



(e) $\epsilon = 8$



(f) $\epsilon = 10$

FIGURE 5.6: Examples of marked CIFAR-100 images under different ϵ.

Table 5.8: The impact of the choice of feature extractor used in marked-data generation on the performance of our method applied on CIFAR-100. ResNet18 and VGG16 were pretrained on ImageNet while CLIP was developed by OpenAI, pretrained on millions image-text pairs from the Internet. "Untrained" refers to using a randomly initialized ResNet18. "Gaussian mark" and "Bernoulli mark" refer to the settings where there is no feature extractor available. ResNet18 (pretrained on ImageNet) was the default in other experiments. All results were averaged over 20 experiments.

	CG		C$\bar{\text{G}}$		$\bar{\text{C}}$G		$\bar{\text{C}}\bar{\text{G}}$	
	TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
ResNet18	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%
VGG16	100%	2.0%	100%	2.2%	100%	5.1%	100%	7.2%
CLIP	100%	2.1%	100%	2.6%	100%	6.5%	100%	6.6%
Untrained	100%	2.8%	100%	3.2%	100%	11.4%	100%	12.6%
Gaussian mark	100%	5.4%	100%	5.6%	100%	26.8%	100%	28.6%
Bernoulli mark	100%	2.7%	100%	3.1%	100%	12.4%	100%	20.8%

Table 5.9: CIFAR-100 auditing results under different upper bounds of false-detection rate (0.05 is the default). We set $\alpha' = \alpha/2$. All results are averaged over 20 experiments.

	CG		C$\bar{\text{G}}$		$\bar{\text{C}}$G		$\bar{\text{C}}\bar{\text{G}}$	
	TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
$\alpha = 0.05$	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%
$\alpha = 0.01$	100%	3.2%	100%	3.6%	100%	8.9%	100%	7.8%
$\alpha = 10^{-3}$	100%	3.9%	100%	4.2%	100%	13.3%	100%	12.1%
$\alpha = 10^{-4}$	100%	6.1%	100%	7.0%	100%	17.0%	100%	18.6%
$\alpha = 10^{-5}$	100%	7.9%	100%	8.6%	100%	19.0%	100%	23.9%
$\alpha = 10^{-6}$	100%	8.9%	100%	9.7%	100%	29.6%	100%	24.3%

Table 5.10: The impact of π on the performance of our method applied on CIFAR-100. $\pi = 16$ was the default in other experiments. cost is the number of queries to the ML model f for detecting data use. That is, cost = RCD $\times\, q \times n \times \pi$. All results were averaged over 20 experiments.

	CG			C$\bar{\text{G}}$			$\bar{\text{C}}$G			$\bar{\text{C}}\bar{\text{G}}$		
	TDR	RCD	cost	TDR	RCD	cost	TDR	RCD	cost	TDR	RCD	cost
$\pi = 1$	100%	4.1%	410	100%	5.0%	495	100%	67.1%	6,702	0%	100.0%	10,000
$\pi = 2$	100%	3.6%	709	100%	4.1%	814	100%	32.1%	6,408	100%	64.0%	12,796
$\pi = 4$	100%	3.4%	1,334	100%	3.7%	1,457	100%	15.3%	6,107	100%	27.7%	11,047
$\pi = 8$	100%	2.6%	2,038	100%	3.1%	2,419	100%	8.2%	6,509	100%	10.8%	8,632
$\pi = 16$	100%	1.9%	2,964	100%	2.0%	3,087	100%	5.9%	9,312	100%	6.0%	9,492
$\pi = 32$	100%	3.0%	9,364	100%	3.2%	9,975	100%	5.1%	16,192	100%	4.8%	15,146

Table 5.11: Robustness of our proposed method on CIFAR-100 against countermeasures. The most effective countermeasure to degrade the detection performance of our method is differential privacy, but it also destroyed the utility of the ML model. All results were averaged over 20 experiments.

		acc%	CG		CḠ		C̄G		C̄Ḡ	
			TDR	RCD	TDR	RCD	TDR	RCD	TDR	RCD
No adaptive attack		74.29	100%	1.9%	100%	2.0%	100%	5.9%	100%	6.0%
Masking output	Top-5	74.29	100%	2.1%	100%	2.1%	-	-	-	-
	Top-1	74.29	100%	2.4%	100%	2.4%	-	-	-	-
	MemGuard	74.29	100%	15.7%	100%	16.5%	-	-	-	-
Memorization reduction	DPSGD($\sigma = 0.001$)	70.01	100%	6.5%	100%	51.7%	100%	8.3%	100%	36.7%
	DPSGD($\sigma = 0.002$)	64.11	100%	39.8%	5%	99.9%	100%	57.4%	10%	99.7%
	DPSGD($\sigma = 0.003$)	59.25	90%	81.4%	0%	100.0%	50%	93.4%	0%	100.0%
	EarlyStop(60)	73.50	100%	2.5%	100%	2.8%	100%	5.1%	100%	6.3%
	EarlyStop(40)	69.15	100%	7.0%	100%	32.7%	100%	14.8%	100%	32.9%
	EarlyStop(20)	67.10	100%	31.8%	5%	100.0%	100%	56.8%	15%	95.1%
	AdvReg	60.18	100%	7.4%	100%	17.8%	100%	9.1%	100%	29.0%
Other attacks	PairDetect	74.29	100%	2.0%	100%	2.0%	100%	6.4%	100%	7.4%
	NoTrainAug	61.59	100%	1.9%	100%	3.0%	100%	5.1%	100%	8.5%
	Rand($\sigma = 10$)	70.64	100%	4.0%	100%	4.5%	100%	17.2%	100%	20.8%
	Rand($\sigma = 20$)	65.97	100%	10.2%	100%	15.6%	100%	53.1%	95%	71.2%
	Rand($\sigma = 30$)	62.10	100%	40.3%	100%	66.4%	80%	89.3%	30%	99.0%
	MarkPerturb	70.49	100%	5.2%	100%	5.6%	100%	30.0%	100%	32.9%

Table 5.12: Results on auditing data in visual encoder trained by SimCLR, under an upper bound of $\alpha = 0.05$ on the false-detection rate. 10% of training samples were marked. All results were averaged over 20 experiments.

	TDR	RCD
CIFAR-10	95%	71.2%
CIFAR-100	100%	72.8%
TinyImageNet	100%	78.2%

(a) Detection performance

(b) Encoder utility

FIGURE 5.7: The impact of epochs on the detection performance and encoder utility. The evaluated encoder was trained by SimCLR on marked CIFAR-100 (10% are marked). The results are averaged over 20 experiments.

Table 5.13: Overall performance of our proposed method on Llama 2 fine-tuned on marked text datasets (10% of fine-tuning samples were marked) for different numbers of epochs, under an upper bound of $\alpha = 0.05$ on the false-detection rate. All results were averaged over 20 experiments.

	SST2			AG's news			TweetEval (emoji)		
	acc%	TDR	RCD	acc%	TDR	RCD	acc%	TDR	RCD
Epoch 0	63.07	0%	100.0%	28.41	0%	100.0%	16.58	0%	100.0%
Epoch 1	95.33	100%	28.7%	91.69	100%	29.7%	40.49	100%	38.9%
Epoch 2	95.26	100%	2.2%	91.68	100%	2.3%	41.88	100%	2.6%
Epoch 3	95.56	100%	1.2%	92.33	100%	1.2%	43.03	100%	1.2%

Table 5.14: Overall performance of our proposed method on CLIP fine-tuned on marked Flickr30k (10% of fine-tuning samples were marked) for different numbers of epochs, under an upper bound of $\alpha = 0.05$ on the false-detection rate. All results were averaged over 20 experiments.

	acc%	TDR	RCD
Epoch 0	80.73	0%	100.0%
Epoch 1	88.44	100%	69.9%
Epoch 2	88.53	100%	23.1%
Epoch 3	88.53	100%	12.1%

Table 5.15: Test accuracies of the audited image classifiers and differences between accuracies of classifiers trained on marked and clean datasets. For CIFAR-100 and TinyImageNet, results are average and standard deviation over 20 classifiers. ImageNet results are for only one model, due to training cost.

	acc(%)	$\triangle$acc
CIFAR-100 (ResNet-18)	$75.61(\pm 0.23)$	$0.08(\pm 0.34)\%$
TinyImageNet (ResNet-18)	$59.86(\pm 0.41)$	$-0.16(\pm 0.60)\%$
ImageNet (ResNet-50)	69.08	-0.05%

Table 5.16: The computation of $\mathsf{MIA}_{A,F'}(x, y, f)$ by membership inference attacks. In RMIA, γ is a hyperparameter that we set $\gamma = 2$ following the previous work [178]. We set $\lambda = 0.3$ for CIFAR-100 and $\lambda = 0.9$ for TinyImageNet. For each $(\acute{x}, \acute{y}) \in A$, $\acute{x}$ denotes an auxiliary data instance and $\acute{y}$ is its associated label. $\left[f(x)\right]_y$ denotes the confidence score of $f(x)$ associated with y.

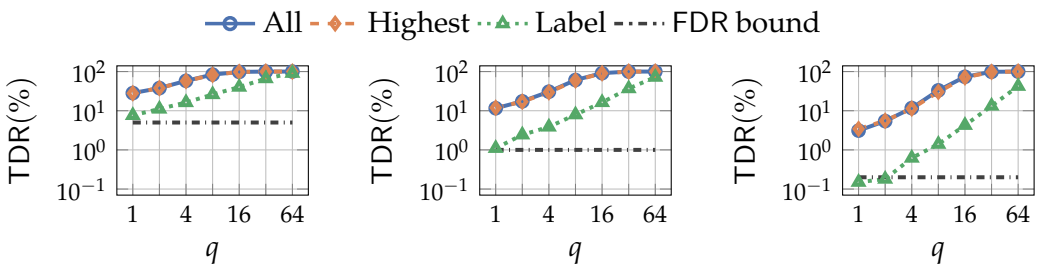
Membership inference attack	$\mathsf{MIA}_{A,F'}(x, y, f)$
Attack-P [171]	$\mathbb{P}_{(\acute{x},\acute{y})\in A}\left(\frac{\left[f(x)\right]_y}{\left[f(\acute{x})\right]_{\acute{y}}} \geq 1\right)$
Attack-R [171]	$\mathbb{P}_{f'\in F'}\left(\frac{\left[f(x)\right]_y}{\left[f'(x)\right]_y} \geq 1\right)$
LiRA [15] (offline)	$1 - \mathbb{P}_{f'\in F'}\left(\log(\frac{\left[f'(x)\right]_y}{1-\left[f'(x)\right]_y}) > \log(\frac{\left[f(x)\right]_y}{1-\left[f(x)\right]_y})\right)$
RMIA [178] (offline)	$\mathbb{P}_{(\acute{x},\acute{y})\in A}\left(\left(\frac{\left[f(x)\right]_y}{\frac{1}{2}\mathsf{avg}_{f'\in F'}((1+\lambda)\left[f(x)\right]_y+(1-\lambda))}\right)\left(\frac{\left[f(\acute{x})\right]_{\acute{y}}}{\frac{1}{2}\mathsf{avg}_{f'\in F'}((1+\lambda)\left[f'(\acute{x})\right]_{\acute{y}}+(1-\lambda))}\right)^{-1} \geq \gamma\right)$

Table 5.17: Limitations of membership inference applied in data-use auditing. "●" means the information is needed while "○" means the information is not needed. "✔" means it provides a bounded FDR while "✗" means it does not.

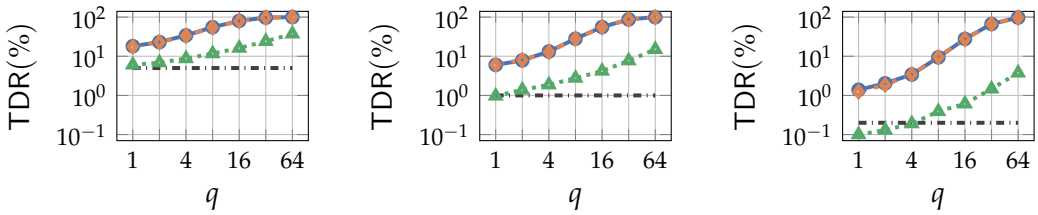
	Auxiliary data	Reference model	Bounded FDR
Our method	○	○	✔
Attack-P [171]	●	○	✗
Attack-R [171]	●	●	✗
LiRA [15]	●	●	✗
RMIA [178]	●	●	✗

Table 5.18: Empirical measures (%) of FDR ($q = 1$) of our data-use auditing method when applied to audit image classifiers that were not trained on the audited data instances. Results are averaged over 500 × 20 detections for CIFAR-100 (1,000 × 20 for TinyImageNet or 1,000 for ImageNet). We trained 20 classifiers for CIFAR-100 or TinyImageNet, and one classifier for ImageNet, in each of which 500 CIFAR-100 (1,000 TinyImageNet or 1,000 ImageNet) training samples were audited. The numbers in the parenthesis are standard deviations among the 20 classifiers.

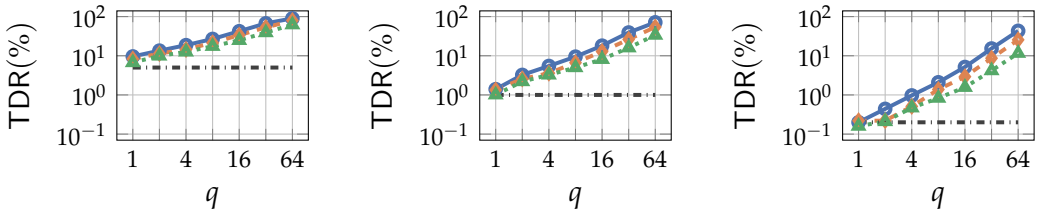
	FDR ≤		
	5%	1%	0.2%
CIFAR-100	4.57(±0.86)	0.86(±0.34)	0.10(±0.10)
TinyImageNet	4.83(±0.64)	0.89(±0.26)	0.07(±0.07)
ImageNet	4.59	0.5	0.0



(a) CIFAR-100: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)

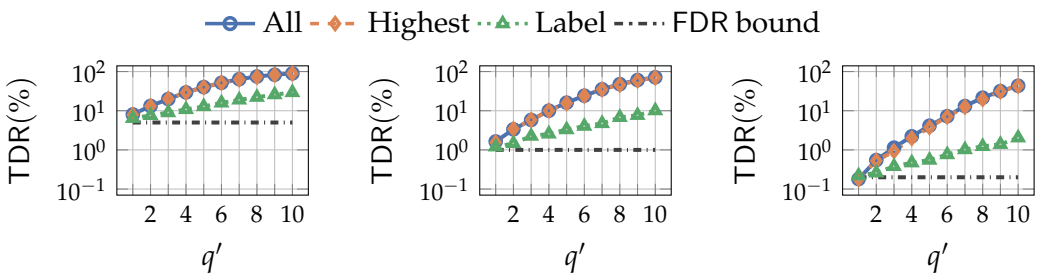


(b) TinyImageNet: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)

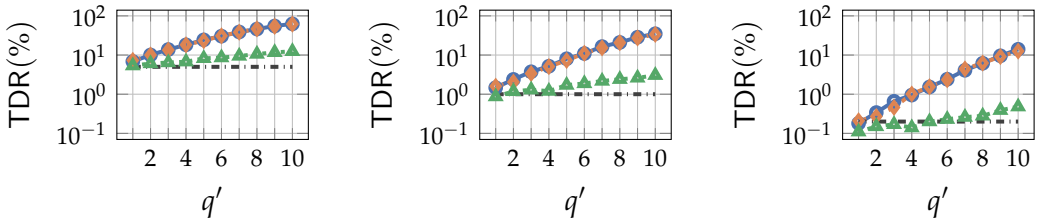


(c) ImageNet: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)

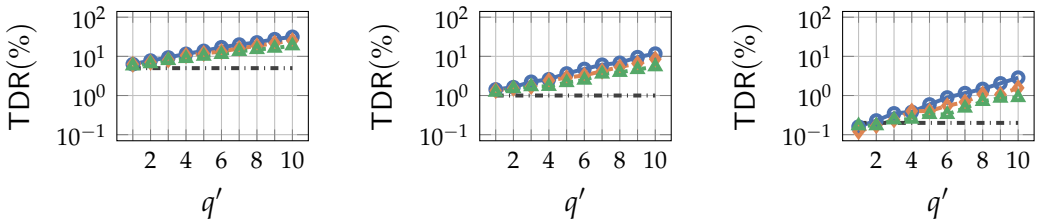
FIGURE 5.8: TDR(%) of auditing image classifiers where a data owner had q data instances to audit and all of them were used in training. "All" means that the output of a classifier is a full vector of confidence scores (our default); "Highest" means that the output is a label and its associated confidence score; "Label" means that the output is a label only.



(a) CIFAR-100: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)

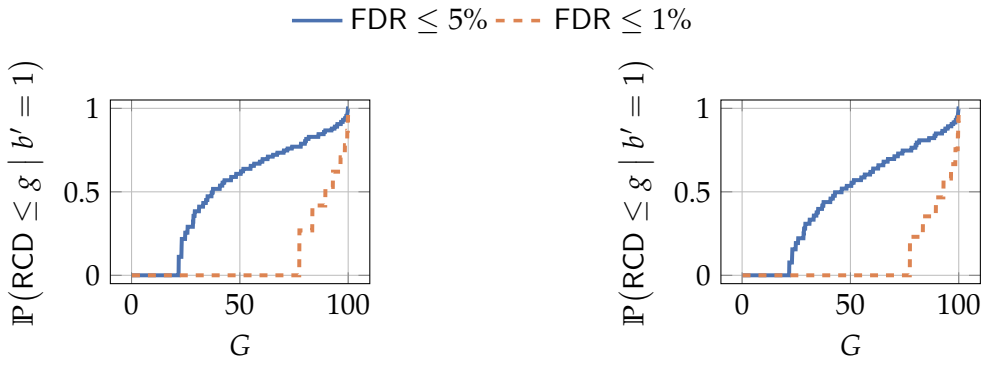


(b) TinyImageNet: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)



(c) ImageNet: @FDR $\leq$ 5% (left), 1% (center), 0.2% (right)

FIGURE 5.9: TDR(%) of auditing image classifiers where a data owner had $q = 10$ data instances to audit and q' of them were used in training. "All" means that the output of a classifier is a full vector of confidence scores (our default); "Highest" means that the output is a label and its associated confidence score; "Label" means that the output is a label only.



(a) CIFAR-100 (b) TinyImageNet

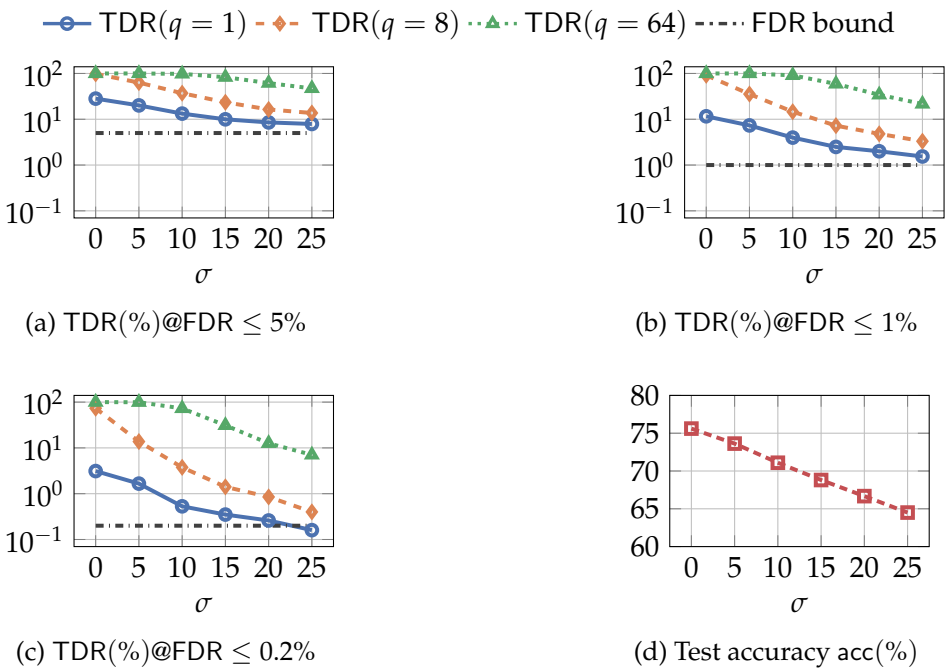
FIGURE 5.10: CDF of RCD in the case $q = 1$ and $n = 1000$, conditioned on data-use being detected.

	FDR ≤	5%				1%				0.2%			
Ours		30.25				12.20				3.25			
Attack-P		7.37				0.92				0.22			
Attack-R	$\acute{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	28.82	30.87	26.40	19.95	14.32	13.05	6.57	3.52	5.72	3.95	1.37	0.75
	2	27.75	28.12	24.17	16.75	10.82	7.65	5.02	2.72	2.60	1.65	0.82	0.45
	3	23.27	23.47	20.10	16.87	3.35	3.54	3.52	3.05	0.52	0.89	0.62	0.72
	4	19.55	20.47	17.55	14.14	2.52	2.97	3.27	2.62	0.50	0.45	0.67	0.52
LiRA	$\acute{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	33.50	28.49	22.30	17.29	14.00	9.95	5.75	4.62	5.57	3.65	1.50	1.10
	2	29.05	25.42	20.27	15.55	9.37	8.10	5.72	4.22	2.97	2.62	1.47	1.07
	3	21.27	20.82	17.67	15.80	4.25	4.40	4.02	3.05	0.92	1.12	0.87	0.77
	4	16.60	18.55	16.50	15.35	4.25	3.75	3.65	3.05	0.95	0.42	0.92	0.57
RMIA	$\acute{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	31.62	30.02	24.22	14.14	16.20	10.40	5.8	2.54	6.67	2.50	0.60	0.00
	2	28.95	29.37	22.40	15.52	10.07	7.07	4.17	3.05	2.87	1.57	0.00	0.67
	3	23.15	23.15	18.67	13.55	4.65	4.25	2.14	2.42	0.95	1.02	0.00	0.00
	4	18.05	18.95	15.05	15.27	4.05	4.42	3.40	2.25	0.65	0.00	0.27	0.00

FIGURE 5.11: Overall comparison of TDR(%) ($q = 1$) across Attack-P, Attack-R, LiRA, and RMIA on CIFAR-100. Results are averaged over 250×20 detections. We trained 20 WideResNet-28-2 classifiers (WideResNet-28-2 is default model architecture in the previous works (e.g., [15])), in each of which 250 training samples of CIFAR-100 were audited. Lighter colors indicate larger improvement of our technique over these membership inference methods.

	FDR $\leq$	5%				1%				0.2%			
Ours		16.06				5.03				0.94			
Attack-P		8.46				1.47				0.33			
Attack-R	$\dot{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	12.52	12.08	10.05	8.11	3.30	3.02	2.35	1.67	1.08	0.62	0.70	0.31
	2	11.75	11.71	9.31	8.11	2.54	2.40	2.06	1.86	0.41	0.47	0.42	0.40
	3	10.27	10.37	8.83	7.76	1.65	1.65	1.72	1.68	0.27	0.27	0.35	0.26
	4	9.22	9.15	8.41	7.96	1.43	1.48	1.65	1.52	0.26	0.20	0.33	0.26
LiRA	$\dot{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	13.60	12.85	10.72	8.59	3.95	2.96	2.38	1.71	0.97	0.58	0.41	0.32
	2	11.22	11.07	9.77	8.66	2.40	1.93	1.60	1.62	0.41	0.37	0.38	0.21
	3	8.72	9.60	9.58	7.78	1.40	1.62	1.46	1.70	0.16	0.37	0.42	0.31
	4	8.08	8.23	8.02	7.71	1.35	1.40	1.47	1.62	0.25	0.25	0.27	0.28
RMIA	$\dot{m}/m =$	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8	1/1	1/2	1/4	1/8
	$\beta = 1$	13.56	13.41	10.67	9.96	4.18	3.26	1.98	2.06	1.41	0.86	0.56	0.11
	2	13.30	11.40	10.50	8.96	3.46	2.38	1.90	1.67	0.53	0.37	0.43	0.37
	3	10.28	10.78	8.66	8.72	1.92	2.18	1.40	1.72	0.33	0.22	0.22	0.50
	4	9.47	9.42	8.69	8.56	1.78	1.90	1.31	1.51	0.26	0.21	0.28	0.42

FIGURE 5.12: Overall comparison of TDR(%) ($q = 1$) across Attack-P, Attack-R, LiRA, and RMIA on TinyImageNet. Results are averaged over 500×20 detections. We trained 20 ResNet-18 classifiers, in each of which 500 training samples of TinyImageNet were audited. Lighter colors indicate larger improvement of our technique over these membership inference methods.



(a) TDR(%)@FDR $\leq$ 5%

(b) TDR(%)@FDR $\leq$ 1%

(c) TDR(%)@FDR $\leq$ 0.2%

(d) Test accuracy acc(%)

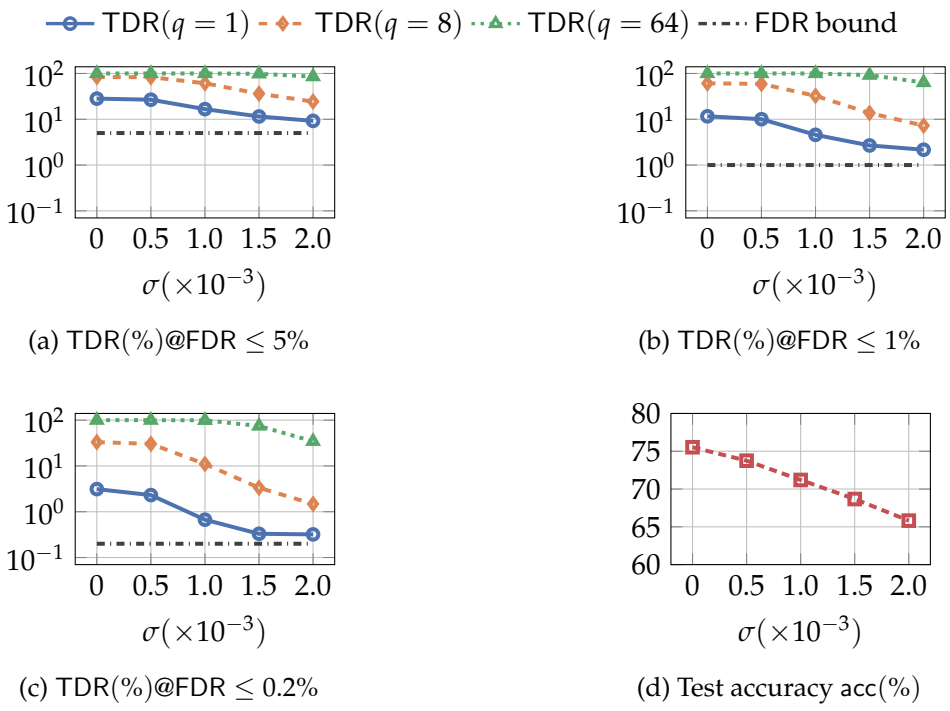
FIGURE 5.13: TDR(%) of our auditing method for CIFAR-100 image classifiers under data perturbation using Gaussian noises (parameterized by standard deviation σ) (Figs. 5.13a–c) and test accuracies of image classifiers (Fig. 5.13d).

Table 5.19: TDR(%) of our method after remarking or image smoothing when applied to audit the use of CIFAR-100 in image classifier (ResNet-18). Results are averaged over 500×20 detections for CIFAR-100. We trained 20 classifiers, in each of which 500 CIFAR-100 training samples were audited.

	acc%	q	FDR $\leq$ 5%	FDR $\leq$ 1%	FDR $\leq$ 0.2%
No perturbation	75.61	1	28.21	11.59	3.11
		16	97.91	91.20	73.48
		64	100.00	100.00	100.00
Remark	71.70	1	12.60	3.73	0.58
		16	35.16	13.75	3.41
		64	97.66	89.54	70.27
Gaussian smooth	39.51	1	7.57	1.48	0.19
		16	13.36	3.33	0.58
		64	46.45	21.22	6.51
Median smooth	57.36	1	9.83	2.39	0.33
		16	24.72	7.99	1.41
		64	85.45	63.04	35.34
General smooth	64.36	1	14.49	3.78	0.68
		16	44.55	19.95	5.71
		64	99.59	97.59	88.22

Table 5.20: TDR(%) ($q = 1$) of our data-use auditing method when applied to audit CIFAR-100 image instances in training image classifiers of different model architectures (ResNet-18 is the default). Results are averaged over 500×20 detections. We trained 20 classifiers, in each of which 500 training samples of CIFAR-100 were audited. The numbers in the parenthesis are standard deviations among the 20 classifiers.

	FDR $\leq$ 5%	FDR $\leq$ 1%	FDR $\leq$ 0.2%
ResNet-18	28.21($\pm$1.60)	11.59($\pm$0.99)	3.11($\pm$0.69)
ResNet-34	26.39($\pm$1.63)	10.52($\pm$0.78)	2.66($\pm$0.41)
WideResNet-28-2	29.10($\pm$1.96)	11.53($\pm$1.43)	2.99($\pm$0.54)
VGG-16	20.80($\pm$1.09)	7.20($\pm$1.16)	1.53($\pm$0.45)
ConvNetBN	29.90($\pm$1.33)	11.89($\pm$1.21)	3.25($\pm$0.69)



(a) TDR(%)@FDR $\leq$ 5%

(b) TDR(%)@FDR $\leq$ 1%

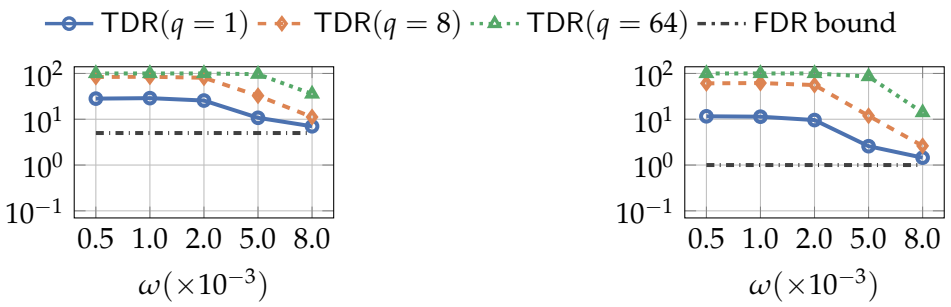
(c) TDR(%)@FDR $\leq$ 0.2%

(d) Test accuracy acc(%)

FIGURE 5.14: TDR(%) of our auditing method for CIFAR-100 image classifiers trained by DPSGD (parameterized by noise multiplier σ) (Figs. 5.14a–c) and test accuracies of image classifiers (Fig. 5.14d).

Table 5.21: TDR(%) ($q = 1$) of our data-use auditing method when applied to audit the use of CIFAR-100 in image classifier (ResNet-18), when we set $n = 200$, $n = 500$, $n = 1{,}000$, and $n = 5{,}000$ ($n = 1{,}000$ is our default). Results are averaged over 500×20 detections for CIFAR-100. We trained 20 classifiers, in each of which 500 CIFAR-100 training samples were audited. The numbers in the parenthesis are standard deviations among the 20 classifiers.

	FDR $\leq$			
	5%	1%	0.2%	0.1%
$n = 200$	26.07($\pm$1.61)	6.96($\pm$1.10)	–	–
$n = 500$	27.41($\pm$1.72)	10.19($\pm$1.29)	–	–
$n = 1{,}000$	28.21($\pm$1.60)	11.59($\pm$0.99)	3.11($\pm$0.69)	–
$n = 5{,}000$	28.16($\pm$2.01)	11.79($\pm$1.61)	4.43($\pm$0.96)	2.01($\pm$0.66)



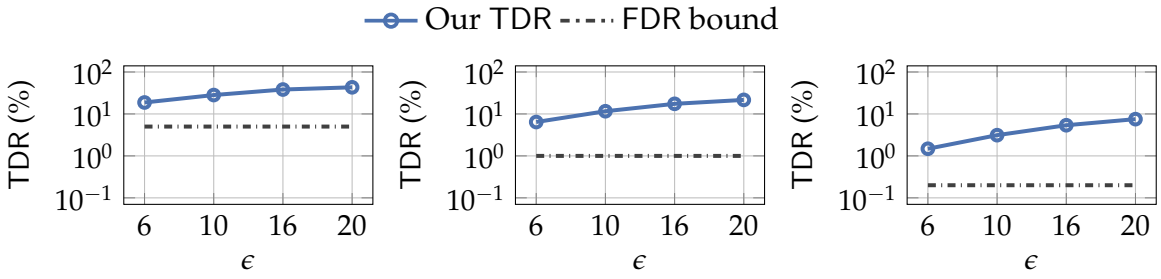
(a) TDR(%)@FDR $\leq$ 5%

(b) TDR(%)@FDR $\leq$ 1%

(c) TDR(%)@FDR $\leq$ 0.2%

(d) Test accuracy acc(%)

FIGURE 5.15: TDR(%) of our auditing method for CIFAR-100 image classifiers trained with varying weight decay (Figs. 5.15a–c) and test accuracies of image classifiers (Fig. 5.15d).



(a) FDR $\leq$ 5%

(b) FDR $\leq$ 1%

(c) FDR $\leq$ 0.2%

FIGURE 5.16: TDR(%) ($q = 1$) of our method for auditing CIFAR-100 instances in image classifiers under varying ϵ values, plotted for three levels of FDR.

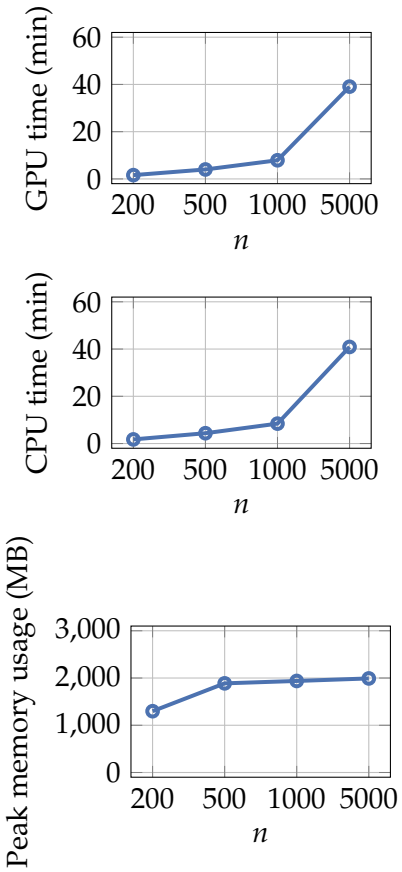


FIGURE 5.17: Overhead of running our data-marking algorithm using a varying n, for each audited data instance ($q = 1$).

Table 5.22: TDR(%) ($q = 1$) of auditing CIFAR-100 instances in image classifiers under different choices of data-marking methods (`OUV+OM` is the default). Results are averaged over 500×20 detections. We trained 20 classifiers, in each of which 500 training samples of CIFAR-100 were audited. The numbers in the parenthesis are standard deviations among the 20 classifiers.

	FDR $\leq$		
	5%	1%	0.2%
`RM`	22.77($\pm$1.58)	8.23($\pm$1.02)	2.10($\pm$0.68)
`RUV+OM`	27.79($\pm$1.47)	11.08($\pm$1.19)	2.97($\pm$0.95)
`OUV+OM`	28.21($\pm$1.60)	11.59($\pm$0.99)	3.11($\pm$0.69)

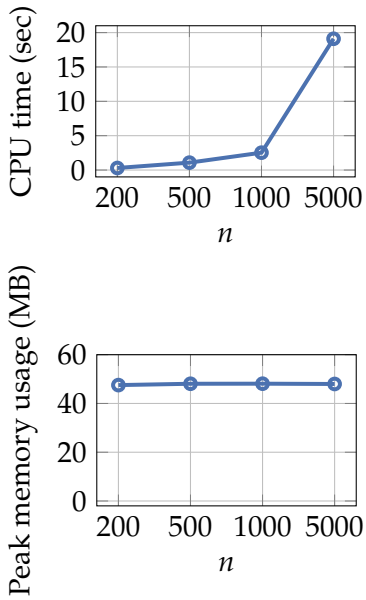
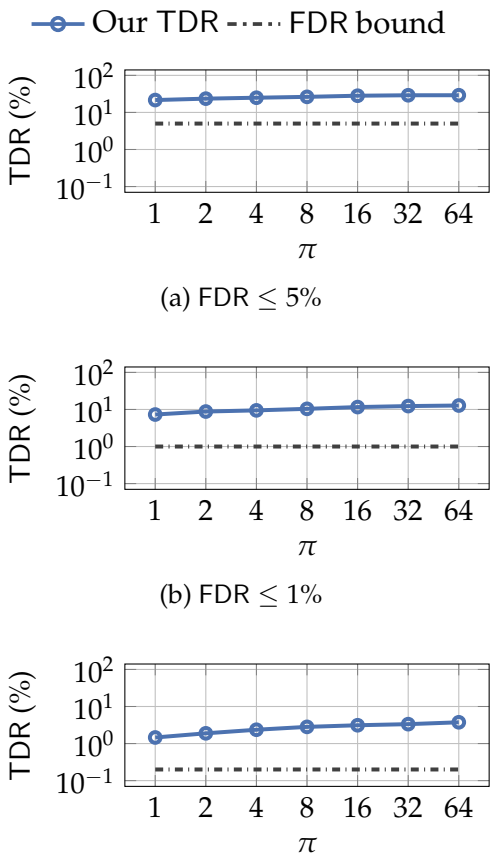


FIGURE 5.18: Overhead of applying prior-posterior-ratio martingale for confidence interval estimation in data-use detection for each audited data instance ($q = 1$), under a varying n.

Table 5.23: TDR(%) ($q = 1$) of the method using the rank of the added mark and our method when applied to audit the use of CIFAR-100 in image classifier (ResNet-18). Results are averaged over 500×20 detections for CIFAR-100. We trained 20 classifiers, in each of which 500 CIFAR-100 training samples were audited.

	FDR $\leq$		
	5%	1%	0.2%
Our method	28.21($\pm$1.60)	11.59($\pm$0.99)	3.11($\pm$0.69)
Use rank of mark	6.21($\pm$4.58)	1.57($\pm$2.28)	0.32($\pm$0.94)

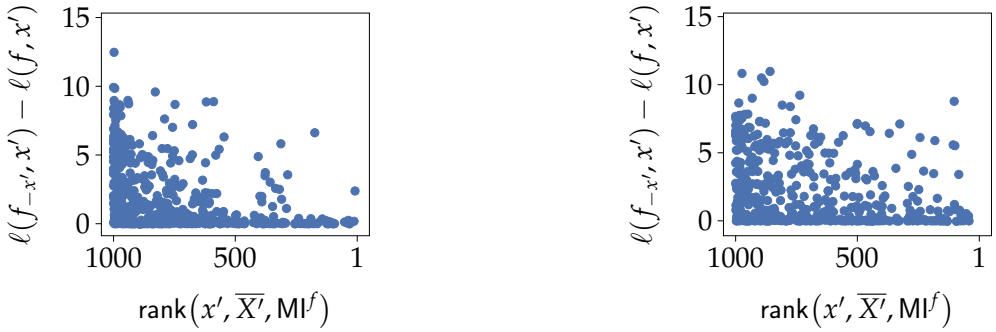


(a) FDR $\leq$ 5%

(b) FDR $\leq$ 1%

(c) FDR $\leq$ 0.2%

FIGURE 5.19: TDR(%) ($q = 1$) of our auditing method for auditing CIFAR-100 instances in image classifiers, using varying π values (π is the number of data augmentations) in data-use detection, plotted for three levels of FDR.



(a) CIFAR-100

(b) TinyImageNet

FIGURE 5.20: $\text{rank}(x', \overline{X'}, \text{MI}^f)$ vs. $\ell(f_{-x'}, x') - \ell(f, x')$

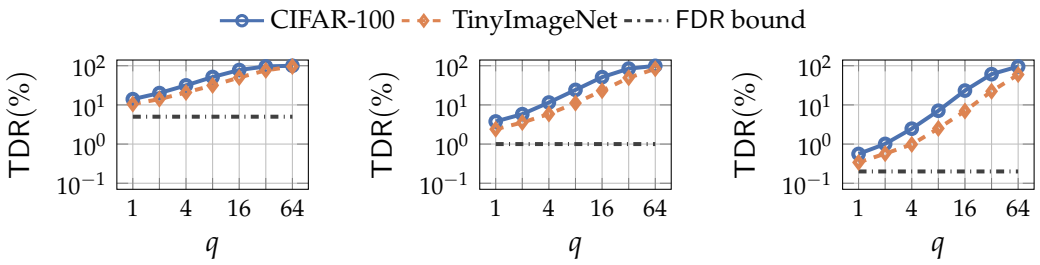


FIGURE 5.21: TDR(%) @FDR $\leq$ 5% (left), 1% (center), 0.2% (right) of auditing visual encoders, when the data owner had q data instances to audit and all of them were used in training.

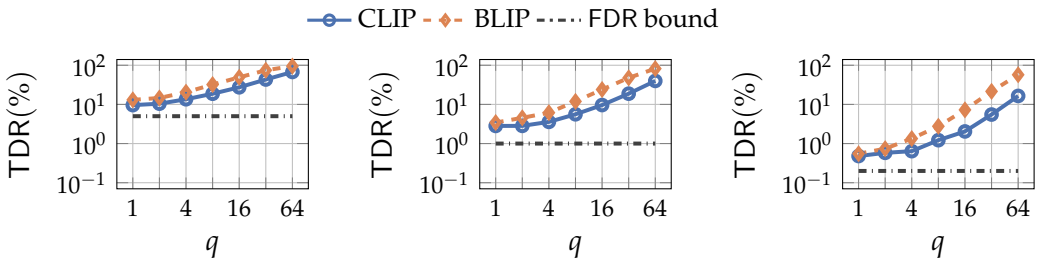
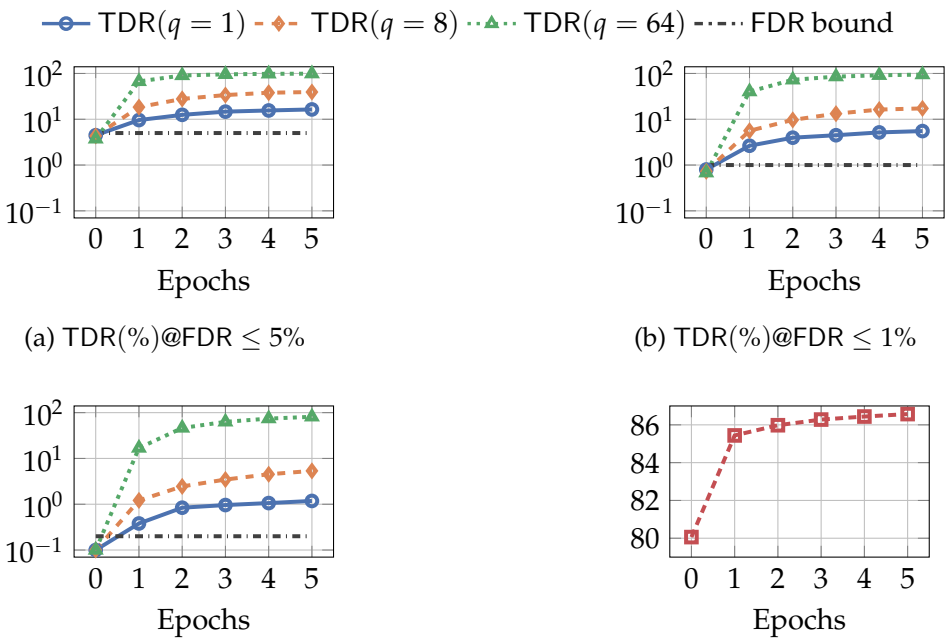


FIGURE 5.22: TDR(%) @FDR $\leq$ 5% (left), 1% (center), 0.2% (right) of auditing the fine-tuned CLIP and BLIP models, when a data owner had q data instances to audit and all of them were used in training.



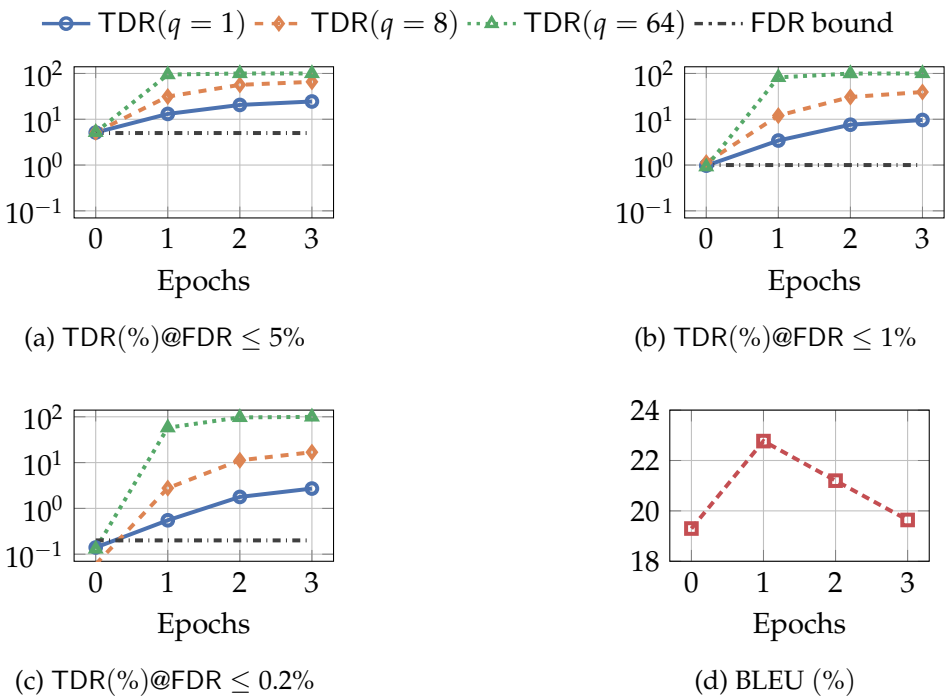
(a) TDR(%)@FDR $\leq$ 5%

(b) TDR(%)@FDR $\leq$ 1%

(c) TDR(%)@FDR $\leq$ 0.2%

(d) Test accuracy acc(%)

FIGURE 5.23: TDR(%) of our auditing method for CLIP fine-tuned on Flickr30k (Figs. 5.23a–c) and test accuracies of fine-tuned CLIP (Fig. 5.23d). Results are averaged over 250 $\times$ 20 detections. We fine-tuned 20 CLIP models, in each of which 250 training samples were audited.



(a) TDR(%)@FDR $\leq$ 5%

(b) TDR(%)@FDR $\leq$ 1%

(c) TDR(%)@FDR $\leq$ 0.2%

(d) BLEU (%)

FIGURE 5.24: TDR(%) of our auditing method for BLIP fine-tuned on Flickr30k (Figs. 5.24a–c) and test accuracies of fine-tuned BLIP (Fig. 5.24d). Results are averaged over 250 $\times$ 20 detections. We fine-tuned 20 BLIP models, in each of which 250 training samples were audited.

6. AcidWash: A Framework to Purify Auditable Training Data

In this chapter, we study the limits of data-use auditing in adversarial settings. To this end, we propose *AcidWash*, a novel and general data purification framework that a data curator or an ML practitioner can leverage to mitigate data-use auditing, i.e., reducing the likelihood that a data owner detects the unauthorized use of her data in ML models, or significantly increasing the cost required to detect such misuse.

Given a marked training dataset, a subset of which is detected as a reference dataset by an existing detector, AcidWash adds carefully crafted perturbations to the inputs in the remaining, untrusted dataset. In doing so, AcidWash aims to achieve three goals: 1) evasion, which is that AcidWash mitigates data-use auditing of a model trained on the purified training data; 2) utility, which is that a model learned on the purified training data is accurate; and 3) generality, which is that AcidWash is applicable to a wide range of data-use auditing algorithms.

To achieve the evasion goal, our key observation is that a model trained on the reference dataset is unlikely to have the auditable property since an overwhelming majority of the reference data are clean (assuming the detector is reasonably accurate). Therefore, AcidWash perturbs data in the untrusted dataset so that they follow a similar distribution to the reference dataset. We quantify the distribution similarity between the two datasets using the well-known *Wasserstein distance* [50, 149], due to its intuitive interpretation (the minimum cost of moving a pile of earth in the shape of one distribution to achieve the shape of the other). To achieve the utility goal, AcidWash aims to bound the perturbations added to the untrusted inputs. Formally, we formulate finding the perturbations as a minimization optimization problem, where the objective function minimizes the Wasserstein distance between the two dataset distributions (quantifying the evasion goal) under the constraint that the perturbation magnitudes are bounded (quantifying the utility goal). AcidWash achieves the generality goal by not depending on any specific data-use auditing algorithm, when formulating and solving the optimization problem.

However, it is hard to solve the optimization problem due to the challenge of computing the Wasserstein distance. To address the challenge, we leverage the Kantorovich-Rubinstein duality [152] to approximate the Wasserstein distance as the maximum difference between the average function value of the untrusted inputs and that of the reference inputs, where the function value of an input is calculated using a neural network that implements a 1-Lipschitz function. Then, we refine our optimization problem as a min-max one, where the inner max problem aims to find the 1-Lipschitz neural network that approximates the Wasserstein distance and the outer min problem aims to find the perturbations. We detail a two-step method to iteratively solve the min-max optimization problem. One step starts from current perturbations and solves the inner max problem to update the neural network used to approximate the Wasserstein distance. The second step starts from the current neural network approximating Wasserstein distance and solves the outer min problem to update the perturbations.

We evaluate AcidWash against two state-of-the-art data-use auditing algorithms in auditing image classifiers. The first is Radioactive Data [118] that we have introduced in Sec. 5.3.1.1, and the second is our general data-use auditing framework proposed in Chapter 5. Both have empirically demonstrated good performance for data-use detection while providing a guarantee on the false-detection rate, a key property for a reliable data-use auditing method [181]. Our results on multiple image benchmark datasets show that AcidWash can effectively mitigate these two data-use auditing methods (i.e., decrease the likelihood that a data owner successfully detects the use of her data to train the ML model or increase the number of model queries needed for a successful detection) while still achieving a highly accurate trained model, even if the reference dataset includes a small fraction of marked data. Moreover, we show that AcidWash outperforms existing data purification methods, notably Friendly Noise [84] and state-of-the-art robust learning methods [3, 47, 184]. That is, when the models trained on the AcidWash-purified data achieve similar accuracy to those trained on unpurified data, those trained on the AcidWash-purified data are more likely to evade data-use detection or require substantially higher query costs to be detected.

To summarize, our key contributions are as follows:

- We propose AcidWash, a general framework to purify auditable training data, designed to mitigate data-use auditing of ML models.
- We formulate data purification as an optimization problem and leverage a two-step method to approximately solve it.
- We show the effectiveness of AcidWash by applying it to mitigate two state-of-the-art data-use auditing techniques, namely Radioactive Data [118] and our data-use auditing framework proposed in Chapter 5.

6.1 Problem Formulation

We consider three entities: a *data owner*, a *data curator*, and a *model trainer*. The data owner holds a set X of data that she wants to publish online (e.g., on social media platforms). The data curator collects data from various sources, e.g., the public Internet, preprocesses them (e.g., labeling the collected data if needed), and then sells the curated dataset to a model trainer. We denote the preprocessed dataset as D. The model trainer aims to develop a useful ML model and he does so by applying an ML algorithm to learn a model on the data from the data curator. Then the model trainer deploys the trained model as a cloud service or an end-user application. In this work, we mainly focus on image data and image classifiers.

Here we assume a data curator because data collection and labeling often require a large amount of data, computation, and human resources, which may not be available to a model trainer. Consequently, the model trainer obtains the training dataset from the data curator. Assuming such a data curator is realistic as there are many of them in the real world, e.g., Argilla (`https://argilla.io`), Cleanlab (`https://cleanlab.ai`), and Lightly (`https://www.lightly.ai`). While we distinguish between a data curator and a model trainer to make our system setup more general, one party could play both roles.

6.1.1 Threat Model

The data curator might collect the published data owned by the data owner *but without her authorization*. In other words, X or its published version might be part of D. Consequently, the training dataset used by the model trainer to train his ML model might contain the data owner's published data, which raises the data owner's concern on unauthorized use of her data in model training. To address this concern, the data owner can apply a data-use auditing method (e.g., [62, 118]) that embeds marks into her data prior to publication and later detects whether her data have been used in a given ML model. In response, the data curator and the model trainer may seek to evade detection of such unauthorized data use even when the training dataset includes the data owner's audited data. To achieve this, the data curator purifies the collected dataset to produce a *purified training dataset*, denoted as $\tilde{D}$, before providing it to the model trainer. The goal of the data curator is to mitigate data-use auditing of an ML model trained on the purified dataset.

Data-use auditing in ML models: Data-use auditing allows the data owner to detect whether her published data have been used in an ML model's training. It consists of a data-marking algorithm mark deployed prior to data publication and data-use detection algorithm detect used after an ML model's deployment. The data-marking algorithm takes as input a raw dataset X, and outputs a marked dataset X' to be published and a secret set $\overline{X'}$ to be hidden:

$$(X', \overline{X'}) \leftarrow \mathsf{mark}(X).$$

The data-use detection algorithm takes as inputs a given ML model f, the published marked dataset X', and the secret set $\overline{X'}$, and it outputs a binary result $b' \in \{0, 1\}$ indicating whether the published data X' has been used in the training of f:

$$b' \leftarrow \mathsf{detect}(f, X', \overline{X'}),$$

where $b' = 1$ indicates that the use of X' in training f is detected while $b' = 0$ indicates that it is not.

Radioactive Data proposed by Sablayrolles et al. [118] is an example of data-use auditing. Specifically, given a raw dataset X, its data-marking algorithm randomly samples a mark per data class and perturbs the data samples in X as X' such that their feature vectors (extracted by a pretrained feature extractor, e.g., ResNet-18 pretrained on ImageNet used in their work) are aligned with the corresponding sampled mark. The secret set $\overline{X'}$ returned by its data-marking algorithm is the set of sampled marks. Then, given an ML model f, its data-use detection algorithm checks whether the weights of its final layer have similar directions with the sampled marks in $\overline{X'}$, by a hypothesis test. It returns $b' = 1$ if the p-value of the hypothesis test is no larger than a threshold α or returns $b' = 0$ if not. Here setting a threshold α guarantees that its false-detection rate (i.e., probability that the detection algorithm returns a detected result when the ML model did not use the published marked data in training) is bounded by α.

We consider Radioactive Data [118] and our auditing framework proposed in Chapter 5 as the data-use auditing methods applied by the data owner. This is because these two methods have empirically demonstrated effectiveness on auditing data use in image classifiers and provide a bounded false-detection rate, a key property for a reliable data-use auditing method as highlighted by previous works (e.g., [62, 181]).

Assumptions on data curator's capabilities: We assume the data curator has a feature extractor $\widetilde{\mathsf{Enc}}$, which outputs a feature vector for an input. For instance, the feature extractor could be pre-trained on ImageNet [34] or a publicly available one pre-trained using self-supervised learning; e.g., OpenAI makes its CLIP feature extractor [114] pre-trained on 400 million image-text pairs publicly available (`https://openai.com/research/clip`). Moreover, we assume the data curator has access to a *reference dataset* R. The data curator could obtain the reference dataset R from a trustworthy source; e.g., its employees could generate R themselves. Alternatively, the data curator can use a subset of D that an automated detector identified as clean as R. R does not need to be completely clean; i.e., the detector can be imperfect. However, we assume that the fraction of marked inputs in R is much smaller than in D. Thus, a model trained based on R alone is less likely to have the auditable

property. Obtaining such a small set of reference data is commonly and reasonably assumed in previous works on defense against data poisoning attacks (e.g., [47, 154, 179]). The data curator uses R as a reference to purify the remaining untrusted data $D \setminus R$ to make $\tilde{D}$.

6.1.2 Design Goals

We aim to design a *data purification framework* for the data curator to purify D to produce $\tilde{D}$. Data purification constitutes an attack to data-use auditing techniques. In other words, when the purified training dataset includes the data owner's audited data, an ML model trained on them mitigates the data-use auditing method. In particular, we aim to achieve the following three design goals:

- Evasion: The *evasion goal* means that a model trained on the purified training dataset $\tilde{D}$ mitigates the data-use auditing method by the data owner (i.e., reduces the likelihood that the unauthorized data-use is detected or substantially increases the cost needed for a detection).
- Utility: The purified training dataset is used to train models by model trainers. A model trainer desires a more accurate model. Therefore, the *utility goal* means that the purified training dataset has good utility, so that a model trained on it is as accurate as possible. In other words, the accuracy of the model trained on the purified training dataset should be comparable to that of a model trained on the original, unpurified dataset.
- Generality: The data-use auditing approach used by the data owner may be unknown to the data curator and model trainer. Therefore, the *generality goal* means that the data purification framework should be agnostic to the data-use auditing method.

In this work, we aim to design a data purification framework that achieves these three goals. We only focus on supervised learning and image purification, and leave purification of other types of data or for other types of machine learning models as future work.

6.2 The AcidWash Framework

To achieve the evasion goal, our data purification framework AcidWash aims to perturb the unpurified data to overwhelm the marks added by the data-use auditing method of the data owner, so it mitigates data-use auditing of a model trained on the purified data. A key challenge is how to perturb the unpurified data to achieve both the evasion and utility goals. For instance, one naive method is to add large perturbations to the unpurified data such that they become random noise. This method can achieve the evasion goal, but it substantially sacrifices the utility of the purified data. To address this challenge, our key observation is that the reference dataset achieves the evasion goal; i.e., a model trained on the reference dataset is unlikely to have the auditable property that a data-use auditing method can detect, and so a dataset that is distributed similarly to the reference dataset is likely to achieve the evasion goal. Based on this observation, AcidWash adds a bounded perturbation (achieving the utility goal) to the unpurified data such that the purified data has a feature distribution similar to the reference data (achieving the evasion goal), where the features of an input are extracted by the data curator's feature extractor $\widetilde{\mathsf{Enc}}$. We consider the feature distribution instead of the raw input distribution because features are high-level summaries of an input and are more likely to be relevant to the auditable property of a model.

Formally, we use the well-known Wasserstein distance to quantify the similarity between two distributions and formulate finding such perturbations to the unpurified data as an optimization problem. However, it is challenging to solve the optimization problem due to the complexity of the Wasserstein distance. To address this challenge, we propose a two-step method to approximately solve the optimization problem, which iteratively alternates between optimizing the perturbations added to the unpurified data and approximating the Wasserstein distance between the feature distributions of the current purified data and reference data. AcidWash achieves the generality goal by not depending on any specific data-use auditing method when formulating the optimization problem and solving it.

6.2.1 Formulating an Optimization Problem

Formulating an optimization problem: Given a marked training dataset $D = \bigcup_{\acute{z}=1}^{c} D_{\acute{z}}$ ($D_{\acute{z}}$ denotes the marked training dataset of class $\acute{z}$), a reference dataset $R = \bigcup_{\acute{z}=1}^{c} R_{\acute{z}}$ that $R_{\acute{z}} \subset D_{\acute{z}}$, and a feature extractor $\widetilde{\mathsf{Enc}}$, we aim to purify the remaining *untrusted dataset* $U = D \setminus R = \bigcup_{\acute{z}=1}^{c} \{x_1^{\acute{z}}, x_2^{\acute{z}}, \cdots, x_{\bar{m}_{\acute{z}}}^{\acute{z}}\}$ where $\{x_1^{\acute{z}}, x_2^{\acute{z}}, \cdots, x_{\bar{m}_{\acute{z}}}^{\acute{z}}\}$ is the untrusted dataset of class $\acute{z}$. Our goal is to add a perturbation $\tilde{\delta}_i^{\acute{z}}$ to each $x_i^{\acute{z}}$ such that the purified untrusted dataset $\tilde{U} = \bigcup_{\acute{z}=1}^{c} \tilde{U}_{\acute{z}}$ ($\tilde{U}_{\acute{z}} = \{x_1^{\acute{z}} + \tilde{\delta}_1^{\acute{z}}, x_2^{\acute{z}} + \tilde{\delta}_2^{\acute{z}}, \cdots, x_{\bar{m}_{\acute{z}}}^{\acute{z}} + \tilde{\delta}_{\bar{m}_{\acute{z}}}^{\acute{z}}\}$) achieves the evasion and utility goals. The purified training dataset $\tilde{D}$ then consists of both R and $\tilde{U}$. For each class $\acute{z}$, we formulate finding the perturbations $\{\tilde{\delta}_i^{\acute{z}}\}_{i=1}^{\bar{m}_{\acute{z}}}$ as the following optimization problem:

$$\min_{\{\tilde{\delta}_i^{\acute{z}}\}_{i=1}^{\bar{m}_{\acute{z}}}} \quad \Delta(\mathbb{U}, \mathbb{Q}) \tag{6.1}$$

$$\text{s.t.} \quad \sum_{i=1}^{\bar{m}_{\acute{z}}} \left\| \tilde{\delta}_i^{\acute{z}} \right\|_\infty \leq \tilde{\epsilon} \tag{6.2}$$

$$x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}} \in [0, 255]^{\mathsf{dim}}, \forall i = 1, 2, \cdots, \bar{m}_{\acute{z}} \tag{6.3}$$

where $\mathbb{U}$ and $\mathbb{Q}$ are respectively the distributions of the purified data and reference data in the feature space defined by the feature extractor $\widetilde{\mathsf{Enc}}$; Δ is a distance metric between two distributions; dim is the number of dimensions of an input/perturbation; and $[0, 255]^{\mathsf{dim}}$ is the domain of an input, e.g., the image domain with pixel values in the range $[0, 255]$. Both distributions $\mathbb{U}$ and $\mathbb{Q}$ are defined in the feature space $\mathcal{V}$. Specifically, if we sample an input x from the purified dataset $\tilde{U}_{\acute{z}}$ (or the reference dataset $R_{\acute{z}}$) uniformly at random, its feature vector $\widetilde{\mathsf{Enc}}(x)$ has a probability $\mathbb{U}(\widetilde{\mathsf{Enc}}(x))$ (or $\mathbb{Q}(\widetilde{\mathsf{Enc}}(x))$) under the distribution $\mathbb{U}$ (or $\mathbb{Q}$).

In the optimization formulation, Eq. (6.1) aims to achieve evasion, while Eq. (6.2) aims to achieve utility. Specifically, Eq. (6.1) aims to perturb the untrusted data such that the purified data has a similar feature distribution with the reference data. Since a model trained on the reference data is likely to achieve the evasion goal, the purified data, whose features follow a similar distribution with the reference data, is also likely to achieve the evasion goal. Eq. (6.2) aims to bound the perturbations added to the untrusted data to preserve its

utility. Our optimization problem essentially achieves a trade-off between the evasion and utility goals, which can be tuned by the hyperparameter $\tilde{\epsilon}$.

Using Wasserstein distance as $\Delta(\mathbb{U}, \mathbb{Q})$: The distance $\Delta(\mathbb{U}, \mathbb{Q})$ is a key component of Acid-Wash. Wasserstein distance offers an interpretable and smooth representation of the distance between two distributions [50]. In particular, Wasserstein distance between two distributions can be interpreted as the minimum cost of moving a pile of earth in the shape of one distribution to the other. Therefore, we choose Wasserstein distance as $\Delta(\mathbb{U}, \mathbb{Q})$. Formally, the Wasserstein distance $\Delta(\mathbb{U}, \mathbb{Q})$ is defined as follows:

$$\Delta(\mathbb{U}, \mathbb{Q}) \triangleq \inf_{\iota \in \Pi(\mathbb{U}, \mathbb{Q})} \mathbb{E}_{(v, \dot{v}) \sim \iota} \left(\|v - \dot{v}\|_2 \right), \tag{6.4}$$

where $\Pi(\mathbb{U}, \mathbb{Q})$ is the set of all joint distributions for a pair of random variables $(v, \dot{v})$ in the feature space, whose marginals are $\mathbb{U}$ and $\mathbb{Q}$, respectively; $\mathbb{E}\,()$ means expectation; and $(v, \dot{v}) \sim \iota$ means that $(v, \dot{v})$ follows the joint distribution ι.

Approximating the Wasserstein distance $\Delta(\mathbb{U}, \mathbb{Q})$: It is a well-known challenge to compute the Wasserstein distance $\Delta(\mathbb{U}, \mathbb{Q})$ exactly. In particular, in our problem, the two distributions do not have analytical forms, making it even more challenging. To address the challenge, we approximate the Wasserstein distance in a computationally efficient way. First, according to the Kantorovich-Rubinstein duality [152], the Wasserstein distance can be represented as follows:

$$\Delta(\mathbb{U}, \mathbb{Q}) = \sup_{\mathsf{critic} \in \mathcal{H}} \mathbb{E}_{v \sim \mathbb{U}} \left(\mathsf{critic}(v) \right) - \mathbb{E}_{\dot{v} \sim \mathbb{Q}} \left(\mathsf{critic}(\dot{v}) \right), \tag{6.5}$$

where $\mathcal{H}$ is the set of all 1-Lipschitz functions in the feature space. A function $\mathsf{critic} : \mathcal{V} \to \mathbb{R}$ is a 1-Lipschitz function [28] if it satisfies:

$$\left\| \mathsf{critic}(v) - \mathsf{critic}(v') \right\|_2 \leq \left\| v - v' \right\|_2$$

for all $v, v' \in \mathcal{V}$. Note that $\mathcal{H}$ includes all possible 1-Lipschitz functions, which makes it still hard to compute $\Delta(\mathbb{U}, \mathbb{Q})$ using Eq. (6.5). To address the challenge, we parameterize a 1-Lipschitz function as a neural network (we use a two-layer neural network in our experiments) and we approximate $\mathcal{H}$ as $\mathcal{H}_\vartheta$ that consists of all 1-Lipschitz two-layer neural networks. We

use a simple two-layer neural network because the input is a vector and this simple model makes our method more efficient. Then, we approximate $\Delta(\mathbb{U}, \mathbb{Q})$ as follows using $\mathcal{H}_\vartheta$:

$$\Delta(\mathbb{U}, \mathbb{Q}) \approx \max_{\mathsf{critic} \in \mathcal{H}_\vartheta} \mathbb{E}_{v \sim \mathbb{U}} \left(\mathsf{critic}(v)\right) - \mathbb{E}_{\dot{v} \sim \mathbb{Q}} \left(\mathsf{critic}(\dot{v})\right).$$

The expectation $\mathbb{E}_{v \sim \mathbb{U}} \left(\mathsf{critic}(v)\right)$ can be further approximated by the average function value of critic for the perturbed data, i.e., $\frac{1}{|\tilde{U}_{\acute{z}}|} \sum_{x \in \tilde{U}_{\acute{z}}} \mathsf{critic}(\widetilde{\mathsf{Enc}}(x))$, which we denote as $\mathsf{avg}(\mathsf{critic}, \tilde{U}_{\acute{z}})$. Similarly, $\mathbb{E}_{\dot{v} \sim \mathbb{Q}} \left(\mathsf{critic}(\dot{v})\right)$ can be approximated as $\mathsf{avg}(\mathsf{critic}, R_{\acute{z}})$. Thus, we have $\Delta(\mathbb{U}, \mathbb{Q}) \approx \max_{\mathsf{critic} \in \mathcal{H}_\vartheta} \mathsf{avg}(\mathsf{critic}, \tilde{U}_{\acute{z}}) - \mathsf{avg}(\mathsf{critic}, R_{\acute{z}})$.

Refining the optimization problem: Now approximating $\Delta(\mathbb{U}, \mathbb{Q})$, we can reformulate Eqs. (6.1)–(6.3) as:

$$\min_{\{\tilde{\delta}_i^{\acute{z}}\}_{i=1}^{\bar{m}_{\acute{z}}}} \max_{\mathsf{critic} \in \mathcal{H}_\vartheta} \mathsf{avg}(\mathsf{critic}, \tilde{U}_{\acute{z}}) - \mathsf{avg}(\mathsf{critic}, R_{\acute{z}}) \tag{6.6}$$

$$\text{s.t.} \sum_{i=1}^{\bar{m}_{\acute{z}}} \left\| \tilde{\delta}_i^{\acute{z}} \right\|_\infty \leq \tilde{\epsilon} \tag{6.7}$$

$$x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}} \in [0, 255]^{\mathsf{dim}}, \forall i = 1, 2, \cdots, \bar{m}_{\acute{z}}. \tag{6.8}$$

Note that the reformulated optimization problem is a min-max one with an inner maximization optimization and an outer min optimization.

6.2.2 Solving the Optimization Problem

It is challenging to exactly solve the min-max optimization problem in Eqs. (6.6)–(6.8) due to the complexity of the neural network critic. To address this challenge, we propose an iterative two-step solution to approximately solve it, which is shown in Alg. 3. Specifically, we first initialize each perturbation $\tilde{\delta}_i^{\acute{z}}$ as 0^{dim} and critic as a random neural network. Then, we iteratively alternate between Step I and Step II, where Step I aims to solve the inner max problem to update critic and Step II aims to solve the outer min problem to update the perturbations.

Step I: Solve the inner max problem: Given the current perturbations, Step I solves the inner max problem to update critic. Formally, the inner max problem is as follows: $\max_{\mathsf{critic} \in \mathcal{H}_\vartheta} \ell(\mathsf{critic}) \triangleq$

Algorithm 3 Our Data Purification Framework AcidWash

Input: A feature extractor $\widetilde{\mathsf{Enc}}$, a reference dataset $R = \bigcup_{\acute{z}=1}^{c}\{\acute{x}_i^{\acute{z}}\}_{i=1,2,\ldots,\acute{m}_{\acute{z}}}$, an untrusted dataset $U = \bigcup_{\acute{z}=1}^{c}\{x_i^{\acute{z}}\}_{i=1,2,\ldots,\bar{m}_{\acute{z}}}$, learning rate l_h for critic, learning rate l_x for perturbations, purification ratio ρ, perturbation bound $\bar{\epsilon}$, purification budget F, amplification scale φ, gradient penalty ν, and batch size batchsize.

for $\acute{z} = 1, 2, \ldots, c$ **do**
  $\tilde{\delta}_i^{\acute{z}} \leftarrow 0^{\mathsf{dim}}, \forall i = 1, 2, \cdots, \bar{m}_{\acute{z}}$; `//Initialize the perturbations`
  critic $\leftarrow$ a randomly initialized two-layer neural network; `//Initialize neural network critic`
  iter $\leftarrow 0$;
  while $\sum_{i=1}^{\bar{m}_{\acute{z}}} \mathbb{I}\left(\left\|\tilde{\delta}_i^{\acute{z}}\right\|_\infty > 0\right) < F$ **do**
    iter $\leftarrow$ iter $+ 1$;
    Set $inner = 50$ when iter $= 1$, and $inner = 5$ when iter > 1;
    `//Step I: Solve the inner max optimization problem to update critic.`
    while $inner > 0$ **do**
      $inner \leftarrow inner - 1$;
      `//Use mini-batches to update critic.`
      Sample a mini-batch $\{i_{\grave{z}}\}_{\grave{z}=1}^{\mathsf{batchsize}}$ of batchsize indices from $\{1, 2, \ldots, \bar{m}_{\acute{z}}\}$;
      Sample a mini-batch $\{i'_{\grave{z}}\}_{\grave{z}=1}^{\mathsf{batchsize}}$ of batchsize indices from $\{1, 2, \ldots, \acute{m}_{\acute{z}}\}$;
      $\ell(\mathsf{critic}) \leftarrow \mathsf{avg}(\mathsf{critic}, \{x_{\acute{z}}^{i_{\grave{z}}} + \tilde{\delta}_{i_{\grave{z}}}^{\acute{z}}\}_{\grave{z}=1}^{\mathsf{batchsize}}) - \mathsf{avg}(\mathsf{critic}, \{\acute{x}_{i'_{\grave{z}}}^{\acute{z}}\}_{\grave{z}=1}^{\mathsf{batchsize}})$;
      $regularizer(\mathsf{critic}) \leftarrow 0$;
      for $\grave{z} = 1, 2, \cdots,$ batchsize **do**
        Sample a number η from the interval [0,1] uniformly at random;
        $v \leftarrow \eta\widetilde{\mathsf{Enc}}(x_{\acute{z}}^{i_{\grave{z}}} + \tilde{\delta}_{i_{\grave{z}}}^{\acute{z}}) + (1-\eta)\widetilde{\mathsf{Enc}}(\acute{x}_{i'_{\grave{z}}}^{\acute{z}})$;
        $regularizer(\mathsf{critic}) \leftarrow regularizer(\mathsf{critic}) + \frac{1}{\mathsf{batchsize}}(\|\nabla_v \mathsf{critic}(v)\|_2 - 1)^2$;
      end for
      critic $\leftarrow \mathsf{Adam}(\ell(\mathsf{critic}) + \nu \cdot regularizer(\mathsf{critic}), \mathsf{critic}, l_h)$; `//Update critic using the Adam optimizer`
    end while
    `//Step II: Solve the outer min optimization problem to update perturbations.`
    Select indices batch $\subseteq \{1, \ldots, \bar{m}_{\acute{z}}\}$, $|\mathsf{batch}| = \lfloor \rho \bar{m}_{\acute{z}} \rfloor$, such that $\forall i \in \mathsf{batch}, \forall i' \notin \mathsf{batch} \Rightarrow \mathsf{critic}(x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}}) \geq \mathsf{critic}(x_{i'}^{\acute{z}} + \tilde{\delta}_{i'}^{\acute{z}})$;
    `//Optimize the perturbation for each selected input.`
    for $i \in \mathsf{batch}$ **do**
      Compute $\ell(\tilde{\delta}_i^{\acute{z}}) = \mathsf{critic}(\widetilde{\mathsf{Enc}}(x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}}))$;
      Update perturbation $\tilde{\delta}_i^{\acute{z}}$ using gradient descent: $\tilde{\delta}_i^{\acute{z}} \leftarrow \tilde{\delta}_i^{\acute{z}} - l_x \cdot \nabla_{\tilde{\delta}_i^{\acute{z}}} \ell(\tilde{\delta}_i^{\acute{z}})$;
      Clip $\tilde{\delta}_i^{\acute{z}}$ such that $\left\|\tilde{\delta}_i^{\acute{z}}\right\|_\infty \leq \bar{\epsilon}$ and $x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}}$ is within the domain $[0, 255]^{\mathsf{dim}}$;
    end for
  end while
  for $i = 1, 2, \cdots, \bar{m}_{\acute{z}}$ **do**
    $\tilde{\delta}_i^{\acute{z}} \leftarrow \varphi \cdot \tilde{\delta}_i^{\acute{z}}$; `//Amplify a perturbation.`
    Clip $\tilde{\delta}_i^{\acute{z}}$ such that $x_i^{\acute{z}} + \tilde{\delta}_i^{\acute{z}}$ is within the domain $[0, 255]^{\mathsf{dim}}$; `//Project into the legitimate domain.`
  end for
end for

Output: $\tilde{D} \leftarrow R + \bigcup_{\acute{z}=1}^{c}\{x_1^{\acute{z}} + \tilde{\delta}_1^{\acute{z}}, x_2^{\acute{z}} + \tilde{\delta}_2^{\acute{z}}, \cdots, x_{\bar{m}_{\acute{z}}}^{\acute{z}} + \tilde{\delta}_{\bar{m}_{\acute{z}}}^{\acute{z}}\}$

$\mathsf{avg}(\mathsf{critic}, \tilde{U}_{\acute{z}}) - \mathsf{avg}(\mathsf{critic}, R_{\acute{z}})$. We use a gradient ascent based method to iteratively solve the max problem. Specifically, in each iteration, we sample a mini-batch of $\tilde{U}_{\acute{z}}$ and a mini-batch of $R_{\acute{z}}$ to estimate $\ell(\mathsf{critic})$; and then we use the Adam optimizer [70] to update critic. However, a key challenge is that critic should be a 1-Lipschitz function, i.e., $\mathsf{critic} \in \mathcal{H}_\vartheta$. The

updated critic in the above iterative process does not necessarily guarantee that critic is a 1-Lipschitz function. To address this challenge, we add a penalty term to the loss function $\ell(\mathsf{critic})$, which is inspired by prior work on approximating Wasserstein distance [50]. The key idea of the penalty term is to reduce the magnitude of the gradient of the function critic with respect to its input, so the updated critic is more likely to be a 1-Lipschitz function. lines 15–20 in Alg. 3 correspond to calculating the penalty term, where ν is a penalty coefficient used to balance $\ell(\mathsf{critic})$ and the penalty term. We repeat the iterative process until the iteration number reaches a predefined threshold *inner*.

Step II: Solve the outer min problem: Given the current critic, Step II solves the outer min problem to update the perturbations. Instead of updating each perturbation $\tilde{\delta}_i^z$, we only update those whose corresponding purified inputs are far away from the centroid of the distribution of the reference data, i.e., have high function values under the current critic. We do this to enhance the utility of the purified data. Specifically, we select the ρ fraction of inputs in the purified dataset that have the largest function values under critic. For each selected purified input $x_i^z + \tilde{\delta}_i^z$, we update its $\tilde{\delta}_i^z$. Specifically, for a perturbation $\tilde{\delta}_i^z$, the inner min problem becomes: $\mathsf{min}_{\tilde{\delta}_i^z}\, \mathsf{critic}(\widetilde{\mathsf{Enc}}(x_i^z + \tilde{\delta}_i^z))$. We use gradient descent to update $\tilde{\delta}_i^z$ for one iteration (Line 28 in Alg. 3) and clip it such that $\left\|\tilde{\delta}_i^z\right\|_\infty$ is bounded by $\bar{\epsilon}$ and each purified input is within the legitimate domain $[0,255]^{\mathsf{dim}}$ (Line 29 in Alg. 3).

Stopping criteria, perturbation amplification, and projection: We alternate between Step I and Step II until the total number of nonzero perturbations reaches a predefined threshold F, i.e., $\sum_{i=1}^{\bar{m}_z} \mathbb{I}\left(\left\|\tilde{\delta}_i^z\right\|_\infty > 0\right) = F$. After stopping the iterative process, we amplify each perturbation by φ (Line 33) to better achieve the evasion goal [21]. As such, we control $\tilde{\epsilon}$ by tuning $\bar{\epsilon}$, F, and φ since $\tilde{\epsilon} = \bar{\epsilon} \cdot F \cdot \varphi$. Finally, we clip the perturbation such that each purified input is within the legitimate domain $[0,255]^{\mathsf{dim}}$ (Line 34 in Alg. 3).

6.3 Experiments

In this section, we implement AcidWash as an attack on two state-of-the-art data-use auditing methods: Radioactive Data [118] and our auditing framework proposed in Chapter 5.

We used the open-source code of Radioactive Data in our experiments.[1]

6.3.1 Experimental Setup

Datasets: We used two visual benchmarks in our experiments: CIFAR-100 [72] and Tiny-ImageNet [75]. Please see their descriptions in Sec. 5.3.1.1.

Marking setting: To construct each marked training dataset, we first randomly sampled ϕ of training samples from a given dataset as a data owner's dataset X. We set $\phi = 10\%$ by default (i.e., 50 samples per class in CIFAR-100 or TinyImageNet were assumed to be owned by a data owner). We also experimented with varying ϕ. Then we applied the data-marking algorithm of a chosen data-use auditing method to generate the marked version X' of X.

Specifically, when applying Radioactive Data, we randomly sampled c independent unit vectors and added perturbation to each image in X such that the inner product between the feature vector of the perturbed/marked image (extracted by a pretrained feature extractor Enc) and the corresponding unit vector was maximized while the ℓ_∞-norm of the added perturbation was bounded by ϵ. The resulting perturbed images formed X' and the c unit vectors formed the secret set $\overline{X'}$.

When applying our auditing framework, we set $n = 2$ and $\epsilon = 10$, and used ResNet-18 pretrained on ImageNet as Enc. We followed the marking setting described in Sec. 5.3.1.1 to generate marked data X'. As such, we combined X' with the remaining training samples to make the marked training dataset D.

Selecting reference data: AcidWash requires a detector to detect a reference dataset R from a marked training dataset D. We consider both *simulated detectors* and *real-world detectors* for a comprehensive evaluation of AcidWash in different scenarios. A simulated detector enables us to explicitly control the performance of the detector and thus the quality of the reference dataset, which makes it possible to understand the performance of AcidWash in different scenarios. A real-world detector is a detector that is currently available to a data curator, which enables us to understand the performance of AcidWash as of now.

[1] `https://github.com/facebookresearch/radioactive_data`

- Simulated detectors: We characterize a simulated detector using its *precision* $\mathsf{precision}$ and *recall* recall. Precision is the fraction of inputs in the detected reference dataset R that are unmarked, while recall is the fraction of unmarked inputs in the marked training dataset D that are included in R. We denote the size of D as m. Given a precision $\mathsf{precision}$ and recall recall, we uniformly sampled a subset of size $\lfloor m \times (1-\phi) \times \mathsf{recall} \times (1-\mathsf{precision})/\mathsf{precision} \rfloor$ from the marked inputs and a subset of size $\lfloor m \times (1-\phi) \times \mathsf{recall} \rfloor$ from the unmarked inputs, and merged them to create R. When the precision of this detector is 1.0, this simulates the scenario where reference data is collected from trusted sources. Note that ϕ of the training data is marked. Therefore, we consider the precision $\mathsf{precision} > 1-\phi$ (i.e., $\mathsf{precision} > 0.9$ in our default setting) so that the simulated detector is better than the random detector that selects the reference data from the marked training dataset uniformly at random.
- Real-world detectors: This type of detector represents an algorithm available as of now that can be used to detect reference data. In our experiments, we considered the k-nearest neighbors (k-NN) detector [111], where k is set as the number of training samples per class, i.e., $k = 500$ for CIFAR-100 or TinyImageNet. The previous work [111] demonstrated its effectiveness in the scenario where an ML model is trained by fine-tuning the linear classifier with its feature extractor frozen. However, it is less effective to remove all the data that introduces hidden properties into an ML model trained from scratch (i.e., it achieves a high recall but a low precision). Therefore, we adapted the k-NN detector such that the data selected by it are mostly unmarked, enabling this data to be used as R in purification. We denote its adapted version as k-NN(ζ) where ζ is number of detected samples per class. k-NN(ζ) works as follows: (1) A feature extractor (e.g., the same feature extractor used for purification) is used to extract features of the inputs in D; (2) A k-NN model is trained on the extracted features with labels; (3) For each class, images are ranked according to each one's confidence score of being the associated label; (4) The top ζ samples are selected as unmarked and returned as members of reference dataset R. We experimented with

different options of $\zeta \in \{10, 25, 50, 100, 150\}$.

Purification setting: We implemented AcidWash to purify each $U = D \setminus R$ to make a purified training dataset $\tilde{D}$. In our experiments, we used two visual encoders, namely `ViT-B-16-SigLIP` and `ViT-B-16`, from OpenCLIP library [26, 64, 114, 123], as $\widetilde{\mathsf{Enc}}$. In other words, we obtained the feature representation of an image by using the two visual encoders to extract its feature vectors and then concatenating the two resulting vectors to form a single feature representation. We followed the previous works (e.g., [50]) to set $\nu = 10$. In addition, we set $\mathsf{batchsize} = 256$; $l_h = 0.001$; $l_x = 0.1$; $\rho = 0.02$; $\bar{\epsilon} = 10$; $F = 100$; and $\varphi = 5$ for CIFAR-100 and $\varphi = 10$ for TinyImageNet. $\bar{\epsilon}$, F, and φ are three hyperparameters impacting the performance of AcidWash. We will study their impacts in Sec. 6.3.2.

Training setting: We trained ResNet-18 models following the training setting described in Sec. 5.3.1.1. When using a purified training dataset to train a model, the detected reference dataset was used as a part of the purified training dataset.

Detection setting: We implemented the data-use detection algorithms of the evaluated data-use auditing methods to determine whether a trained model had used the data owner's audited data. For Radioactive Data, we assumed a white-box access to the model and conducted a hypothesis test to test whether the weights of the final layer of the model are correlated with the marks. The detection result is set as $b' = 1$ if its p-value is no larger than α and $b' = 0$ otherwise. For our auditing method, we assumed a black-box access to the model and queried it using pairs of published marked data and unpublished data in a sequential manner. We assumed the output of the model is a full vector of confidence scores and the ground-truth label of the query is known, which provides the most information to the data owner in data-use detection. We compared membership inference scores of published and unpublished data, and set $b' = 1$ if the estimated number of pairs in which published data yields a higher score exceeds a predefined threshold χ, and $b' = 0$ otherwise. We adopted the same membership inference method introduced in Sec. 5.3.1.1: each input

image was augmented 16 times via random cropping, the outputs of these augmented images were averaged, and the entropy of the averaged output vector was used as its membership inference score. We set α and χ to ensure that the false-detection rate (FDR) of the evaluated data-use auditing methods was provably bounded by 5%.

Evaluation metrics: We used *accuracy* (acc), *true-detection rate* (TDR), and *relative cost for detection* (RCD).

- acc: acc is the fraction of testing samples that are correctly classified by a model and thus it measures the utility of a model.
- TDR: TDR is the fraction of experiments that return a detection result of $b' = 1$ (i.e., the data owner successfully detects the use of her data to train the model).
- RCD: Given that the total number of *possible* queries is $|X| \times 2 \times 16$, RCD is the average percentage of that total that is queried to detect data use. As introduced in Chapter 5, it is used to measure the detection efficiency of the proposed method, which depends on the number of model queries and thus RCD.

A higher acc, a lower TDR, and a higher RCD indicate a better performance on mitigating data-use auditing.

Baselines: We consider the following methods as baselines:

- Reference: The first method, denoted as Ref, is to simply remove the remaining untrusted data after detecting the reference data. That is, models are trained using the reference data R alone.
- Purification: *Random noise*, denoted as $\mathsf{Rand}(\sigma)$, constructs perturbations sampled from a Gaussian distribution with a standard deviation of σ; and *Friendly Noise* [84], denoted as $\mathsf{FriendN}(\varsigma)$, constructs a perturbation that is maximized within a ball of radius ς in the infinity norm but that, when added to the data, does not substantially change the model's output.
- Others: 1) Differentially private stochastic gradient descent (DPSGD) [3] with noise magnitude σ; 2) regularization controlled by the weight decay (denoted as ω) used in the SGD optimizer; 3) Friendly Noise with Bernoulli Noise (FNBN) introduced

into the training process [84], where its noise magnitude is denoted by ς; 4) ASD [47] and CBD [184], two robust learning algorithms originally developed as defense against backdoor attacks; and 5) three image denoising methods, namely Gaussian smoothing, median smoothing, and general smoothing. DPSGD and regularization have been introduced as countermeasures to our proposed auditing framework in Sec. 5.3.1.4. We describe the remaining baselines below.

FNBN [84]: FNBN adds friendly noise of magnitude ς to all the training data before model training and introduces Bernoulli noise sampled from $\{-\varsigma, \varsigma\}$ to augment training samples during model training. As such, FNBN is a combination of purification and robust learning. FNBN is considered as the state-of-the-art defense method against clean-label data poisoning attacks [48, 138]. We followed the previous work [84] to implement this method, where we set $\varsigma \in \{10, 16\}$ and the other hyperparameters to the default values.

ASD [47]: ASD is a robust learning algorithm that dynamically splits samples from the training set to learn. It includes 3 steps: at the first step when the number of epochs is less than 60, starting from 10 samples per class as the clean set (that we used k-NN(ζ) detector with $\zeta = 10$ to select), for every 5 epochs, it moves 10 samples with the lowest symmetric cross-entropy loss per class from the remaining training set to the clean set and leaves the remaining as the polluted set; at the second step when the number of epochs is less than 90, it adds 50% samples of the entire training set with the lowest loss into the clean set and leaves the remaining as the polluted set; at the third step when the number of epochs is less than 120, it applies a meta-split method to detect the hard clean samples from the polluted set. While the clean set and polluted set are dynamically changed, the model is updated by undergoing supervised training on the clean set and semi-supervised training on the polluted set. We followed the previous work to implement it.[2] We set the hyperparameters to the default values.

CBD [184]: CBD is a learning algorithm inspired by causal inference [112]. It firstly trained

[2] `https://github.com/KuofengGao/ASD`

a backdoored model on the poisoned dataset for 5 epochs. Then a clean model is trained for 80 epochs by reweighting the training samples such that the clean model is independent of the backdoored model in the hidden space. We followed the previous work to implement CBD, where we set the hyperparameters to the default values.[3]

6.3.2 Experimental Results

Table 6.1: Average acc and TDR from the models trained on the dataset audited by Radioactive Data, and average acc, RCD, and TDR from the models trained on the dataset audited by our auditing method. The numbers in the parenthesis are standard deviations among the 20 experiment trials.

	Radioactive Data [118]		Our auditing method		
	acc%	TDR@FDR $\leq$ 5%	acc%	RCD(%)@FDR $\leq$ 5%	TDR@FDR $\leq$ 5%
CIFAR-100	74.29($\pm$0.27)	11/20	74.30($\pm$0.27)	2.19	20/20
TinyImageNet	58.97($\pm$0.38)	17/20	59.05($\pm$0.39)	1.29	20/20

Before purification: Table 6.1 shows the results when the marked training dataset was not purified. Our results confirm that the two data-use auditing methods enable a data owner to detect unauthorized data use in model training.

Training models on reference data alone: We present the results of training models on a small set of reference data alone, in Fig. 6.1, Fig. 6.2, Fig. 6.3, and Fig. 6.4. These figures and those presented later that use cell shading indicate results more desirable for the model trainer (i.e., higher acc, lower TDR, or higher RCD) with lighter/less shading. As shown in these figures, TDR by Radioactive Data was close to 0/20 and RCD by our auditing method was close to 100%, when a simulated detector with a high precision precision or a small recall recall was used, or a k-NN(ζ) detector was used. This empirically confirms our hypothesis in Sec. 6.2 that the reference dataset achieves the evasion goal (i.e., the model trained on the reference dataset alone is free of hidden properties detectable by a chosen data-use auditing method). However, acc of the models trained on reference data that we show in these figures were much lower than those trained on the full training dataset (shown

[3] `https://github.com/zaixizhang/CBD`

		acc(%)					TDR				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	5.38	7.10	6.49	6.73	5.45	0.05	0.00	0.05	0.05	0.05
	0.02	2.67	3.41	3.38	3.05	2.61	0.00	0.00	0.05	0.00	0.05
	0.05	20.17	20.18	19.18	19.61	19.25	0.00	0.00	0.00	0.00	0.10
	0.10	32.72	31.98	31.67	31.23	32.72	0.00	0.00	0.05	0.00	0.00
	0.20	49.14	49.07	48.42	47.24	47.78	0.00	0.00	0.15	0.00	0.00
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	3.87	2.84	3.05	6.98	4.17	0.00	0.00	0.10	0.10	0.05
	0.02	4.37	4.11	4.26	4.34	4.17	0.05	0.15	0.05	0.10	0.05
	0.05	12.30	11.74	11.95	11.68	11.46	0.00	0.20	0.05	0.10	0.10
	0.10	23.38	22.99	22.22	22.29	22.06	0.10	0.05	0.10	0.05	0.10
	0.20	35.03	34.56	34.31	33.96	33.78	0.20	0.20	0.20	0.05	0.00

FIGURE 6.1: acc(%) and TDR@FDR $\leq$ 5% of Radioactive Data, from models trained on reference data alone, using a simulated detector with precision precision and recall recall.

		acc(%)					TDR				
CIFAR-100	ζ =	10	25	50	100	150	10	25	50	100	150
		14.73	23.27	29.24	44.87	52.80	0.05	0.00	0.00	0.05	0.00
TinyImageNet	ζ =	10	25	50	100	150	10	25	50	100	150
		8.77	18.15	30.02	40.73	46.50	0.00	0.10	0.05	0.00	0.00

FIGURE 6.2: acc(%) and TDR@FDR $\leq$ 5% of Radioactive Data, from models trained on reference data alone, using a k-NN(ζ) detector.

in Table 6.1). Therefore, using a small set of reference data alone to train models does not satisfy the utility goal.

Purification against Radioactive Data: We compare AcidWash with two purification methods, namely Rand and FriendN, on mitigating Radioactive Data. There exists a trade-off in data purification between the accuracy acc of the trained model and evading the data-use detection. Therefore, to compare AcidWash with those baselines fairly, we searched the perturbation magnitude $\sigma \in [1, \infty)$ and $\varsigma \in [1, \infty)$ in Rand and FriendN such that their acc were as close as possible to acc of AcidWash. We present their acc results in Fig. 6.5 and Fig. 6.6. In Fig. 6.5 and Fig. 6.6, we observe that when the simulated detector had a higher recall recall or the k-NN(ζ) detector had a higher ζ, the model trained on a purified dataset

		acc(%)					RCD(%)				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	5.46	5.45	5.93	5.68	5.58	100.0	100.0	100.0	100.0	100.0
	0.02	2.70	2.82	2.81	2.67	2.92	100.0	100.0	100.0	100.0	100.0
	0.05	20.14	20.06	19.49	19.4	18.96	100.0	100.0	100.0	100.0	100.0
	0.10	32.63	32.41	31.70	31.48	32.58	100.0	100.0	100.0	100.0	100.0
	0.20	49.51	48.52	48.35	47.20	48.06	99.63	95.03	100.0	100.0	100.0
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	3.11	3.04	3.06	3.05	3.04	100.0	100.0	100.0	100.0	100.0
	0.02	4.17	4.09	3.82	4.34	4.33	100.0	100.0	100.0	100.0	95.00
	0.05	12.25	12.05	11.84	11.81	11.77	95.00	100.0	100.0	100.0	100.0
	0.10	23.19	22.83	22.38	22.14	22.09	100.0	100.0	100.0	100.0	100.0
	0.20	34.98	34.58	34.33	33.77	33.85	84.15	97.56	99.97	100.0	94.74

FIGURE 6.3: acc(%) and RCD(%)@FDR $\leq$ 5% of our auditing method, from models trained on reference data alone, using a simulated detector with precision precision and recall recall.

		acc(%)					RCD(%)				
CIFAR-100	ζ =	10	25	50	100	150	10	25	50	100	150
		15.79	25.48	33.77	49.30	56.94	100.0	100.0	100.0	100.0	100.0
TinyImageNet	ζ =	10	25	50	100	150	10	25	50	100	150
		8.60	18.16	29.92	40.90	46.56	100.0	100.0	100.0	100.0	100.0

FIGURE 6.4: acc(%) and RCD(%)@FDR $\leq$ 5% of our auditing method, from models trained on reference data alone, using a k-NN(ζ) detector.

by AcidWash had a higher acc.

Fig. 6.7 shows the detection results when a simulated detector with varying precision and recall was used to select the reference data. AcidWash achieved a plausible trade-off between reducing TDR and maintaining acc. Specifically, for CIFAR-100, when precision = 0.98 and recall = 0.05, AcidWash reduced TDR from 11/20 to 1/20 at FDR $\leq$ 5% while average acc of models trained on its purified dataset was 72.46%, 1.83 percentage points lower than that obtained from the unpurified dataset. For TinyImageNet, under the same detector configuration (i.e., precision = 0.98 and recall = 0.05), AcidWash reduced TDR from 17/20 to 7/20 at FDR $\leq$ 5% while acc of models trained on its purified dataset was 57.54%, 1.43 percentage points lower than that trained on the unpurified dataset. AcidWash consistently

		Rand					FriendN					AcidWash				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	71.94	71.72	71.65	71.92	71.68	71.81	71.86	71.83	71.98	71.82	71.98	71.74	71.92	71.87	71.87
	0.02	72.03	72.03	71.85	71.81	71.80	71.98	71.91	71.90	71.88	71.85	72.20	72.14	72.19	72.07	72.14
	0.05	72.03	71.99	72.04	72.16	72.12	72.14	72.14	72.22	71.97	72.01	72.24	72.30	72.19	72.46	72.41
	0.10	72.22	72.34	72.24	72.36	72.39	72.30	72.23	72.25	72.33	72.25	72.41	72.36	72.46	72.54	72.62
	0.20	72.60	72.63	72.59	72.53	72.63	72.60	72.65	72.54	72.68	72.53	72.40	72.57	72.53	72.72	72.72
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	57.13	57.05	57.05	56.92	57.11	57.28	57.20	57.11	57.10	57.01	57.23	57.34	57.34	57.32	57.24
	0.02	57.15	57.22	57.14	57.18	57.35	57.31	57.30	57.25	57.27	57.22	57.48	57.45	57.54	57.27	57.37
	0.05	57.47	57.61	57.53	57.45	57.49	57.44	57.55	57.58	57.61	57.47	57.43	57.52	57.66	57.54	57.65
	0.10	57.73	57.73	57.77	57.81	57.74	57.75	57.91	57.83	57.67	57.72	57.72	57.82	57.79	57.78	57.79
	0.20	57.83	57.72	57.63	57.57	57.61	57.69	57.65	57.68	57.76	57.78	57.87	57.97	57.94	58.01	58.04

FIGURE 6.5: acc(%) of Radioactive Data, from models trained on purified datasets, using a simulated detector with precision precision and recall recall.

		Rand					FriendN					AcidWash				
CIFAR-100	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		72.81	72.79	72.86	73.10	73.33	72.76	72.88	72.95	72.98	73.02	72.68	72.86	72.89	73.05	73.00
TinyImageNet	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		58.20	58.37	58.36	58.37	58.05	58.44	58.40	58.35	58.35	58.01	58.30	58.41	58.51	58.39	58.23

FIGURE 6.6: acc(%) of Radioactive Data, from models trained on purified datasets, using a k-NN(ζ) detector.

outperformed both Rand and FriendN across all simulated detectors and both datasets. In other words, AcidWash achieved a lower TDR against Radioactive Data than two purification baselines while maintaining comparable acc.

Fig. 6.8 shows the detection results on CIFAR-100 and TinyImageNet in real-world settings where a detection algorithm available as of now, namely k-NN(ζ), was used to select the reference data. In these settings, AcidWash again demonstrated a plausible trade-off between reducing TDR and maintaining acc. Specifically, when $\zeta = 50$, AcidWash reduced TDR from 11/20 to 1/20 at FDR $\leq 5\%$ for CIFAR-100 and the corresponding acc of trained models dropped by 1.41 percentage points. AcidWash reduced TDR from 17/20 to 1/20 at FDR $\leq 5\%$ for TinyImageNet while acc decreased marginally by 0.46 percentage points. Across all evaluated k-NN(ζ) detectors and both datasets, AcidWash consistently outperformed both Rand and FriendN.

To better evaluate the performance of AcidWash in real-world settings, we compare its performance under the k-NN(ζ) detector and that under the simulated detector, ensuring

		Rand					FriendN					AcidWash				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	0.40	0.35	0.45	0.45	0.45	0.35	0.40	0.40	0.45	0.50	0.25	0.15	0.15	0.10	0.25
	0.02	0.45	0.40	0.45	0.50	0.40	0.40	0.45	0.40	0.35	0.40	0.15	0.10	0.05	0.05	0.15
	0.05	0.35	0.40	0.35	0.40	0.45	0.40	0.45	0.45	0.40	0.50	0.15	0.20	0.20	0.05	0.25
	0.10	0.40	0.45	0.50	0.45	0.35	0.50	0.45	0.40	0.35	0.45	0.20	0.25	0.05	0.20	0.10
	0.20	0.45	0.50	0.35	0.45	0.45	0.35	0.50	0.45	0.50	0.45	0.10	0.20	0.05	0.10	0.20
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	0.55	0.55	0.50	0.55	0.50	0.55	0.55	0.55	0.55	0.55	0.45	0.35	0.40	0.35	0.40
	0.02	0.55	0.55	0.50	0.50	0.45	0.55	0.45	0.45	0.45	0.45	0.40	0.40	0.40	0.30	0.30
	0.05	0.45	0.60	0.65	0.45	0.45	0.50	0.55	0.55	0.50	0.50	0.35	0.40	0.40	0.35	0.35
	0.10	0.50	0.50	0.50	0.65	0.50	0.45	0.60	0.55	0.55	0.40	0.30	0.45	0.30	0.35	0.30
	0.20	0.55	0.50	0.60	0.55	0.45	0.45	0.55	0.50	0.55	0.45	0.45	0.35	0.30	0.30	0.20

FIGURE 6.7: TDR@FDR $\leq$ 5% of Radioactive Data, from models trained on purified datasets, using a simulated detector with precision precision and recall recall.

		Rand					FriendN					AcidWash				
CIFAR-100	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		0.45	0.45	0.55	0.50	0.55	0.45	0.40	0.60	0.55	0.60	0.00	0.15	0.05	0.10	0.10
TinyImageNet	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		0.65	0.45	0.70	0.70	0.60	0.55	0.60	0.55	0.60	0.50	0.10	0.05	0.05	0.10	0.05

FIGURE 6.8: TDR@FDR $\leq$ 5% of Radioactive Data, from models trained on purified datasets, using a k-NN(ζ) detector.

that both detectors had the same precision and recall. The comparison results are shown in Fig. 6.9, where the values below each column header represent their precisions and recalls. As in Fig. 6.9, acc of trained models after applying AcidWash using a k-NN(ζ) detector was higher than that from using a simulated detector. In addition, for CIFAR-100, AcidWash achieved a comparable TDR under both detectors. Whereas for TinyImageNet, AcidWash achieved a lower TDR when a k-NN(ζ) detector was applied. Overall, AcidWash performed better when using the real-world detector than the simulated one. This improvement arose because the real-world detector selected data with higher confidence scores as reference data, providing stronger and more informative references for AcidWash to purify the untrusted data.

Purification against our auditing method: We compare AcidWash with the two purification baselines, against our auditing method. We also searched the perturbation magnitude $\sigma \in [1, \infty)$ and $\varsigma \in [1, \infty)$ in Rand and FriendN such that their acc were as close as possible to acc of AcidWash. We present their acc results in Fig. 6.10 and Fig. 6.11.

		acc(%)					TDR				
CIFAR-100	precision =	0.991	0.992	0.991	0.989	0.986	0.991	0.992	0.991	0.989	0.986
	recall =	0.022	0.055	0.110	0.219	0.328	0.022	0.055	0.110	0.219	0.328
	AcidWash (simulated)	72.20	72.45	72.64	72.68	72.79	0.05	0.15	0.05	0.05	0.10
	AcidWash (k-NN(ζ))	72.68	72.86	72.89	73.05	73.00	0.00	0.15	0.05	0.10	0.10
TinyImageNet	precision =	0.972	0.971	0.969	0.965	0.961	0.972	0.971	0.969	0.965	0.961
	recall =	0.021	0.053	0.107	0.214	0.320	0.021	0.053	0.107	0.214	0.320
	AcidWash (simulated)	57.52	57.69	57.70	57.93	58.00	0.35	0.30	0.35	0.25	0.30
	AcidWash (k-NN(ζ))	58.30	58.41	58.51	58.39	58.23	0.10	0.05	0.05	0.10	0.05

FIGURE 6.9: acc(%) and TDR of models trained on the purified datasets, audited by Radioactive Data, using simulated detectors or k-NN(ζ) detectors.

		Rand					FriendN					AcidWash				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	72.04	72.10	71.97	72.04	72.24	72.00	72.15	72.01	72.14	72.12	71.89	71.88	71.97	72.03	71.91
	0.02	72.21	72.12	72.20	72.00	72.15	72.16	72.16	72.18	72.04	72.18	72.15	72.22	72.27	72.21	72.20
	0.05	72.22	72.28	72.24	72.28	72.18	72.34	72.36	72.27	72.41	72.15	72.25	72.25	72.40	72.51	72.49
	0.10	72.49	72.70	72.38	72.40	72.53	72.31	72.35	72.56	72.44	72.40	72.37	72.50	72.59	72.54	72.64
	0.20	72.64	72.67	72.78	72.56	72.63	72.76	72.70	72.87	72.65	72.71	72.37	72.47	72.78	72.72	72.85
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	57.44	57.27	57.33	57.37	57.36	57.41	57.41	57.45	57.44	57.36	57.42	57.34	57.27	57.35	57.25
	0.02	57.43	57.47	57.43	57.55	57.48	57.59	57.49	57.45	57.56	57.53	57.42	57.43	57.53	57.46	57.48
	0.05	57.68	57.60	57.83	57.77	57.65	57.71	57.77	57.67	57.74	57.77	57.53	57.59	57.64	57.65	57.77
	0.10	58.02	57.91	57.89	58.13	57.87	57.93	57.94	58.04	57.87	58.07	57.72	57.74	57.85	57.77	57.81
	0.20	57.97	57.90	57.73	57.69	57.77	57.84	57.81	57.76	57.76	57.80	57.99	57.88	57.87	58.04	58.00

FIGURE 6.10: acc(%) of our auditing method, from models trained on purified datasets, using a simulated detector with precision precision and recall recall.

Fig. 6.12 shows the detection results when a simulated detector was used to select the reference data. AcidWash substantially increased RCD. Specifically, for CIFAR-100, when precision $= 0.98$ and recall $= 0.10$, AcidWash increased RCD from 2.19% to 58.83% at FDR $\leq$ 5% while acc of models trained on its purified dataset was 72.54%, 1.76 percentage points lower than that from the unpurified dataset. For TinyImageNet, under the same detector configuration (i.e., precision $= 0.98$ and recall $= 0.10$), AcidWash increased RCD from 1.29% to 11.85% at FDR $\leq 5\%$ while acc of models trained on its purified dataset was 57.77%, 1.28 percentage points lower than that trained on the unpurified dataset. As precision and recall increased, the performance of AcidWash further improved, yielding a higher RCD. Note that these results came from the detection settings where the audited classifier output full confidence vectors. Under a less informative setting (e.g., the classifier outputs labels only), AcidWash can further mitigate our auditing method. AcidWash consistently outperformed

		Rand					FriendN					AcidWash				
CIFAR-100	$\zeta =$	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		72.99	72.98	73.01	73.22	73.28	72.84	73.05	73.02	73.25	73.26	72.91	72.93	72.88	73.07	73.13
TinyImageNet	$\zeta =$	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		58.76	58.81	58.97	58.62	58.77	58.93	58.89	58.90	58.65	58.68	58.84	58.81	58.93	58.65	58.77

FIGURE 6.11: acc(%) of our auditing method, from models trained on purified datasets, using a k-NN(ζ) detector.

both Rand and FriendN across all simulated detectors and both datasets. In other words, AcidWash achieved a higher RCD than the two purification baselines while maintaining comparable acc.

Fig. 6.13 demonstrates that AcidWash substantially increased RCD in real-world settings where a k-NN(ζ) detector was used to select the reference data. Specifically, when $\zeta = 50$, AcidWash increased RCD from 2.19% to 58.87% at FDR $\leq$ 5% for CIFAR-100 and the corresponding acc of trained models dropped by 1.45 percentage points. Whereas for TinyImageNet, AcidWash increased RCD from 1.29% to 12.01% at FDR $\leq$ 5% and the corresponding acc decreased slightly by 0.12 percentage points only. Across all evaluated k-NN(ζ) detectors and both datasets, AcidWash consistently outperformed both Rand and FriendN, yielding a much higher RCD while maintaining comparable acc.

We further compare the performance of AcidWash under the k-NN(ζ) detector and the simulated detector when both achieved the same precision and recall, to more comprehensively evaluate AcidWash. The comparison results are shown in Fig. 6.14. We observe trends similar to those in the purification experiments against Radioactive Data. Models purified by AcidWash using the k-NN(ζ) detector achieved higher acc than those purified using the simulated detector. Moreover, AcidWash with the k-NN(ζ) detector yielded higher RCD values than using the simulated detector. Overall, AcidWash demonstrated better performance when employing the real-world detector to select reference data, consistent with the conclusion from the Radioactive Data experiments.

Ablation study: We study the impact of φ, F, and $\bar{\epsilon}$ on the performance of AcidWash by varying one hyperparameter while keeping the others fixed. In this study, we used the

		Rand					FriendN					AcidWash				
CIFAR-100	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	2.98	3.16	3.11	2.29	3.22	2.76	2.78	3.00	2.79	2.34	18.75	15.94	16.99	16.31	17.65
	0.02	3.27	3.10	2.64	3.07	3.57	2.29	3.26	3.73	2.62	3.65	24.79	32.72	31.80	29.15	27.73
	0.05	2.91	2.81	3.28	3.25	3.49	2.79	2.65	3.45	2.90	3.17	30.78	31.38	43.50	45.26	42.12
	0.10	3.58	3.54	2.51	2.67	3.24	2.73	3.15	3.19	2.82	3.22	26.00	32.25	47.83	58.83	52.25
	0.20	2.91	3.49	2.62	3.60	2.99	3.11	3.23	3.94	3.30	3.34	24.71	35.10	37.01	60.66	64.54
TinyImageNet	precision =	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0	0.92	0.94	0.96	0.98	1.0
	recall = 0.01	1.95	1.72	1.69	2.07	1.90	2.29	1.54	2.59	1.76	1.77	3.40	3.80	4.11	3.54	4.41
	0.02	1.83	1.63	2.16	1.78	2.28	2.09	2.17	1.89	1.86	1.47	5.84	4.46	5.42	4.78	5.20
	0.05	2.24	1.66	2.09	1.48	2.22	1.70	1.67	1.98	1.58	2.10	5.08	4.13	7.45	7.51	7.64
	0.10	1.28	1.92	2.18	1.71	1.60	1.59	1.56	2.03	1.95	2.04	6.37	5.78	8.47	11.85	10.84
	0.20	1.68	2.13	2.18	2.50	3.26	2.31	2.10	2.33	2.73	2.87	7.61	9.71	9.94	15.11	16.37

FIGURE 6.12: RCD(%)@FDR $\leq$ 5% of our auditing method, from models trained on purified datasets, using a simulated detector with precision precision and recall recall.

		Rand					FriendN					AcidWash				
CIFAR-100	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		2.67	2.43	2.51	2.40	3.47	2.85	2.41	2.58	2.84	2.79	52.51	51.39	58.87	67.18	75.20
TinyImageNet	ζ =	10	25	50	100	150	10	25	50	100	150	10	25	50	100	150
		1.29	1.44	1.13	1.41	1.28	1.58	1.35	1.32	1.41	1.07	10.01	10.09	12.01	13.46	12.13

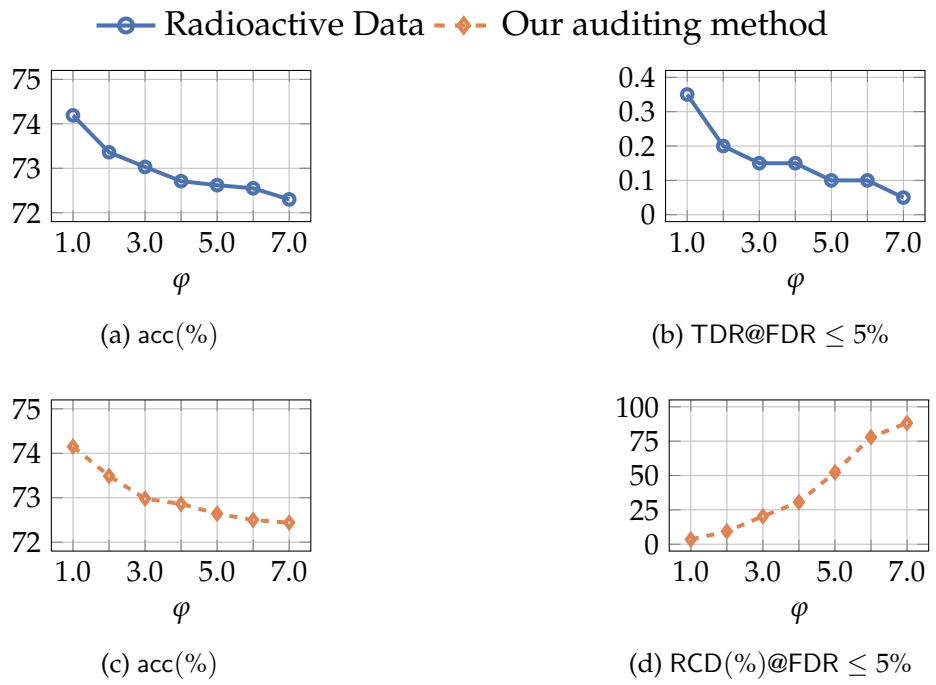
FIGURE 6.13: RCD(%)@FDR $\leq$ 5% of our auditing method, from models trained on purified datasets, using a k-NN(ζ) detector.

simulated detector with precision = 1.0 and recall = 0.1 to select the reference data for AcidWash. We present the results in Fig. 6.15, Fig. 6.16, and Fig. 6.17. When we increased either φ, F, or $\bar{\epsilon}$, AcidWash achieved a better performance on mitigating the data-use auditing method (i.e., achieving a lower TDR against Radioactive Data or a higher RCD against our auditing method) but at the cost of reducing acc. Setting a larger φ, F, or $\bar{\epsilon}$ allows AcidWash to add more perturbation into the training data. Therefore, it reduces the likelihood of the data-use detection or increases the cost needed for a successful detection but sacrificing more data utility. In practice, a data curator can tune φ, F, and $\bar{\epsilon}$ according to the model trainer's requirements, e.g., selecting their values to satisfy specific data/model utility constraints.

Varying the percentage of audited data: We experimented with varying the percentage ϕ of audited data in the training dataset. In these experiments, we used the simulated detector

		acc(%)					RCD(%)				
CIFAR-100	precision =	0.996	0.996	0.995	0.993	0.990	0.996	0.996	0.995	0.993	0.990
	recall =	0.022	0.055	0.110	0.220	0.330	0.022	0.055	0.110	0.220	0.330
	AcidWash (simulated)	72.24	72.45	72.68	72.70	72.84	24.39	44.24	52.48	67.36	69.37
	AcidWash (k-NN(ζ))	72.91	72.93	72.88	73.07	73.13	52.51	51.39	58.87	67.18	75.20
TinyImageNet	precision =	0.966	0.963	0.960	0.955	0.951	0.966	0.963	0.960	0.955	0.951
	recall =	0.021	0.053	0.106	0.212	0.317	0.021	0.053	0.106	0.212	0.317
	AcidWash (simulated)	57.52	57.69	57.70	57.93	58.00	5.60	6.50	7.27	7.84	10.91
	AcidWash (k-NN(ζ))	58.30	58.41	58.51	58.39	58.23	10.01	10.09	12.01	13.46	12.13

FIGURE 6.14: acc(%) and RCD(%) of models trained on the purified datasets, audited by our auditing method, using simulated detectors or k-NN(ζ) detectors.



(a) acc(%)

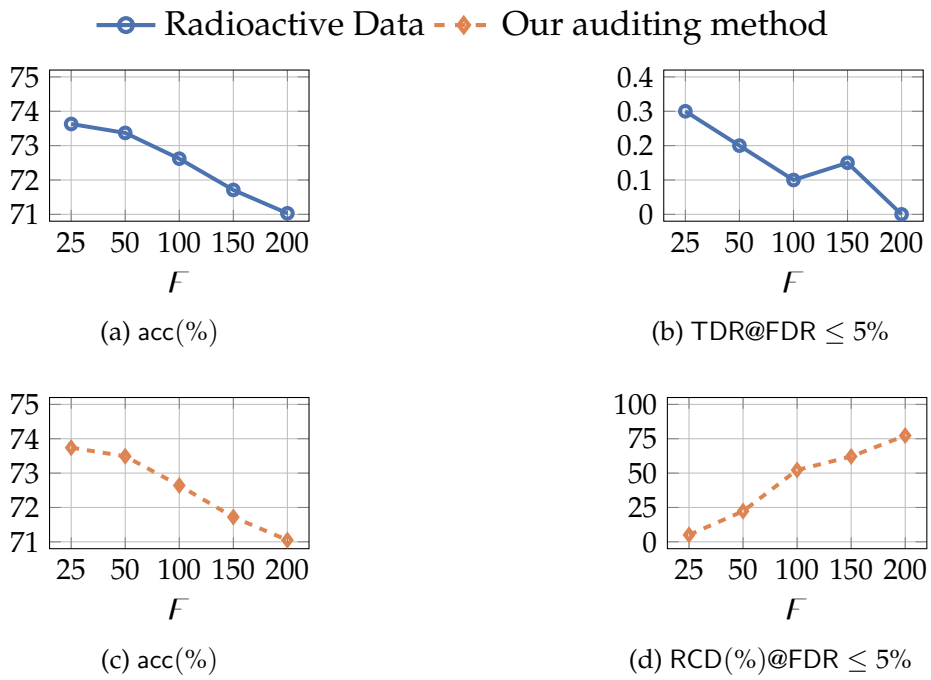
(b) TDR@FDR $\leq$ 5%

(c) acc(%)

(d) RCD(%)@FDR $\leq$ 5%

FIGURE 6.15: The impact of φ on the performance of AcidWash on CIFAR-100 ($\varphi = 5.0$ is our default).

with precision $= 1.0$ and recall $= 0.1$ to select the reference data for AcidWash. We present the results in Fig. 6.18. For Radioactive Data, increasing ϕ yielded a higher TDR but slightly decreased acc when no purification was applied. AcidWash effectively reduced the detection performance of Radioactive Data even when ϕ was high (e.g., $\phi = 30\%$). Specifically, when $\phi = 30\%$, AcidWash decreased its TDR from 20/20 to 11/20 while reducing acc by less than one percentage point. For our auditing method, increasing ϕ did not substantially affect



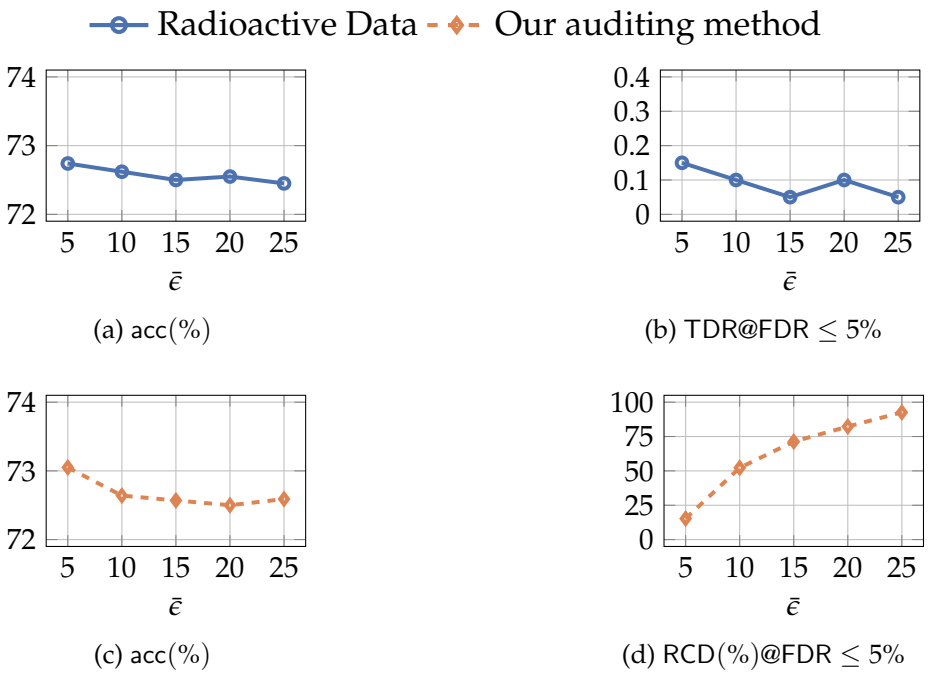
(a) acc(%) (b) TDR@FDR $\leq$ 5%

(c) acc(%) (d) RCD(%)@FDR $\leq$ 5%

FIGURE 6.16: The impact of F on the performance of AcidWash on CIFAR-100 ($F = 100$ is our default).

RCD, but it improved its robustness against AcidWash. In particular, the increase in RCD by AcidWash was only 14.99 percentage points when $\phi = 30\%$, compared to a 50.06-percentage-point increase when $\phi = 10\%$. When ϕ is high, the data curator can correspondingly tune the hyperparameters of AcidWash (e.g., increasing F) to further degrade the detection performance of data-use auditing.

Comparison with other baselines: We compare the performance of AcidWash with other baselines, namely DPSGD, regularization, FNBN, ASD, CBD, and smoothing, as summarized in Table 6.2. To ensure a fair comparison under realistic conditions, we used the k-NN(ζ) detector (with $\zeta = 10$) to select reference data for AcidWash and ASD. As shown in Table 6.2, AcidWash achieved the best overall performance in mitigating both Radioactive Data and our auditing method among all evaluated attacks. Specifically, AcidWash achieved the lowest TDR against Radioactive Data and the highest RCD against our auditing method, while maintaining acc comparable to the highest one.



(a) acc(%) (b) TDR@FDR $\leq$ 5%

(c) acc(%) (d) RCD(%)@FDR $\leq$ 5%

FIGURE 6.17: The impact of $\bar{\epsilon}$ on the performance of AcidWash on CIFAR-100 ($\bar{\epsilon} = 10$ is our default).

6.4 Discussion

6.4.1 How to Design More Robust Data-Use Auditing Techniques

AcidWash works as an attack to mitigate data-use auditing methods. To design a more robust data-use auditing method against AcidWash, the distribution of its marked data should be made indistinguishable from that of the clean data. However, enforcing such indistinguishability in data marking might compromise the auditing method's detection effectiveness. In other words, it might reduce the likelihood that an ML model learns the auditable property that can be detected by the data-use auditing method. Achieving an optimal balance between robustness and effectiveness therefore remains an important direction for future research.

6.4.2 Limitations

AcidWash will fail when a large majority of the collected data are marked (i.e., a large majority of data owners apply data-use auditing methods independently) because a useful

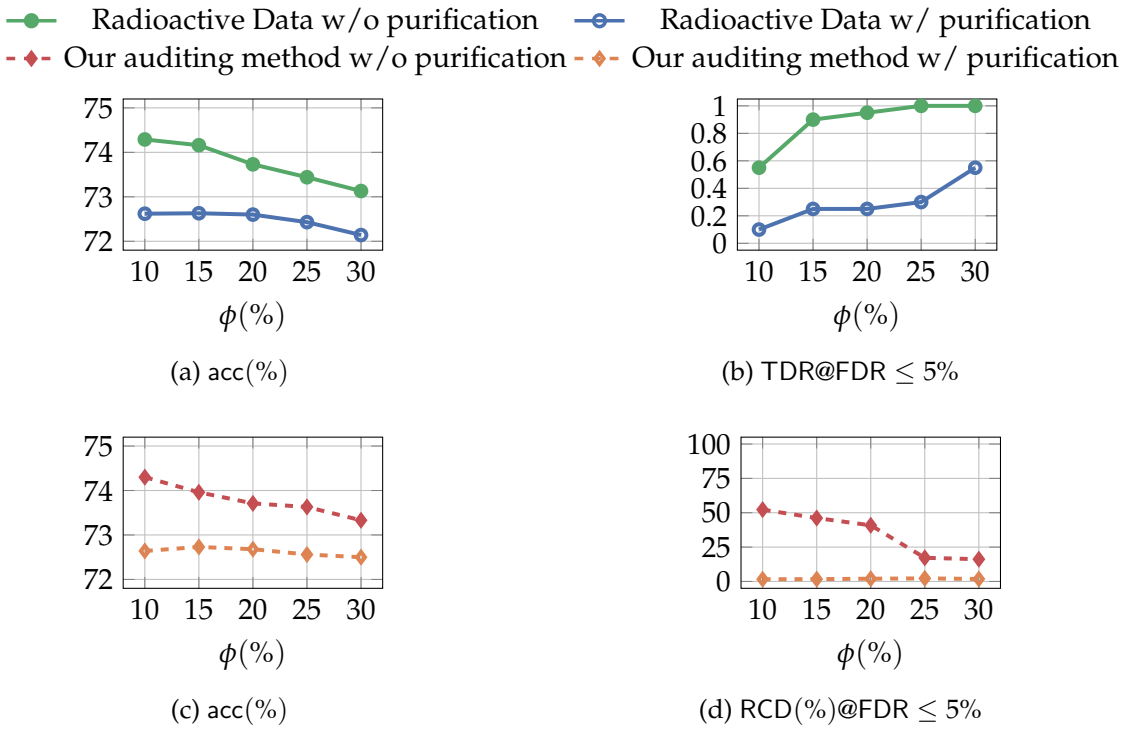


FIGURE 6.18: The performance of AcidWash on CIFAR-100 under different ϕ ($\phi = 10\%$ is our default).

reference dataset cannot be constructed from the collected data (e.g., by an automated detector). In such a scenario, the reference dataset could be constructed from an additional, trustworthy source; e.g., the data curator could generate the reference dataset itself. Once it has a reference dataset where most of its samples are clean, the data curator could apply AcidWash to purify the collected marked data effectively.

6.5 Chapter Summary

In this chapter, we propose AcidWash, a general framework to purify training data. We find that, via adding carefully crafted perturbations to the potentially marked training inputs, we can mitigate data-use auditing methods, while retaining the accuracy of the learned model. Given a reference dataset, finding such perturbations can be formulated as a min-max optimization problem, which can be approximately solved by a two-step method. AcidWash achieves good performance as long as the detector used to detect the

Table 6.2: Comparison between baselines and AcidWash. The numbers in the parenthesis are standard deviations among the 20 experiment trials. In each acc column, we bold the largest number and those within 0.5%, indicating that those methods achieved comparable acc; in TDR@FDR $\leq 5\%$ column, we bold the smallest number(s); and in RCD@FDR $\leq 5\%$ column, we bold the largest number(s). A method with acc, TDR, and RCD bolded means that it achieved the best performance on evading detection when achieving comparable acc with others.

	Radioactive Data [118]		Our auditing method	
	acc%	TDR@FDR $\leq 5\%$	acc%	RCD(%)@FDR $\leq 5\%$
W/o attack	74.29($\pm$0.27)	11/20	74.30($\pm$0.27)	2.19
DPSGD ($\sigma = 0.5 \times 10^{-3}$)	**72.64**($\pm$**0.20**)	7/20	**72.66**($\pm$**0.31**)	3.02
DPSGD ($\sigma = 1.0 \times 10^{-3}$)	69.95($\pm$0.38)	7/20	69.93($\pm$0.32)	7.75
DPSGD ($\sigma = 1.5 \times 10^{-3}$)	67.09($\pm$0.39)	6/20	67.16($\pm$0.29)	15.57
Regularization ($\omega = 3 \times 10^{-3}$)	**72.50**($\pm$**0.24**)	7/20	**72.49**($\pm$**0.27**)	2.77
Regularization ($\omega = 4 \times 10^{-3}$)	70.83($\pm$0.37)	4/20	70.81($\pm$0.38)	7.45
Regularization ($\omega = 5 \times 10^{-3}$)	68.63($\pm$0.31)	4/20	68.53($\pm$0.37)	16.09
FNBN ($\varsigma = 10$)	68.89($\pm$0.21)	5/20	68.72($\pm$0.23)	7.56
FNBN ($\varsigma = 16$)	65.10($\pm$0.32)	2/20	65.37($\pm$0.28)	21.24
ASD	70.84($\pm$0.50)	2/20	70.84($\pm$0.36)	3.89
CBD	**72.69**($\pm$**0.39**)	5/20	**72.61**($\pm$**0.49**)	1.59
Gaussian smoothing	62.08($\pm$1.07)	9/20	62.59($\pm$1.19)	7.58
Median smoothing	55.89($\pm$0.91)	6/20	55.94($\pm$0.64)	20.83
General smoothing	63.39($\pm$0.63)	11/20	63.37($\pm$0.62)	7.38
AcidWash (k-NN($\zeta = 10$))	**72.68**($\pm$**0.27**)	**0/20**	**72.91**($\pm$**0.21**)	**52.51**

reference data provides high precision, though it need not be perfect. We demonstrated the effectiveness and generality of AcidWash by attacking two representative data-use auditing methods, including Radioactive Data and our auditing framework proposed in Chapter 5.

7. Conclusion and Future Work

7.1 Conclusion

This dissertation studies how data owners can detect unauthorized uses of their data in computing systems. As modern digital infrastructures increasingly rely on large datasets from credential databases used in authentication systems to training datasets used in machine learning models, the misuse of such data can lead to serious security, privacy, and economic consequences. Such misuse may occur through credential-database compromises in authentication systems or through the incorporation of datasets into ML model training without the consent of data owners. This dissertation argues that such unauthorized data use can be reliably detected by proactively introducing carefully designed randomness into data and combining anytime-valid statistical tests with system-aware threat modeling, enabling tunable and provably bounded global false-detection guarantees. In addition, this dissertation shows that these detection mechanisms can be mitigated somewhat using additional data, either from other sites in the case of credential-database compromise or from other data sources in the case of ML model training.

First, this dissertation revisits the security of existing honeyword-based credential-database breach detection methods in scenarios where attackers possess passwords leaked from other websites (Chapter 3). Our empirical analysis demonstrates that existing honeyword-generation methods exhibit unfavorable trade-offs between false positives and false negatives when users choose passwords manually. In particular, these approaches suffer from high false-negative rates while also being prone to false alarms. We further examine scenarios in which user passwords are algorithmically generated, such as by password managers. Our results show that existing honeyword-generation techniques provide only limited protection against attackers in this setting. We then explore the use of algorithmic password generators to generate honeywords for accounts protected by password managers and find that the most effective strategy is to generate honeywords using the same password-generation algorithm employed by the user, assuming the defender can identify the generator. Overall, these findings reveal important limitations of current honeyword techniques and highlight new challenges and

opportunities for improving the robustness of credential-breach detection mechanisms.

Building on insights from this analysis, Chapter 4 proposes the first anytime-valid honeyword framework, LeakSentinel, that enables sites to detect credential-database breaches while guaranteeing a tunable and provably bounded global false-detection rate. LeakSentinel consists of a honeyword-generation algorithm compatible with arbitrary password generative models and an online breach-detection algorithm that leverages sequential statistical testing to analyze sequences of incorrect login attempts. Experimental evaluations validate its theoretical false-detection guarantees empirically and demonstrate that LeakSentinel effectively identifies credential-database compromises.

Next, this dissertation addresses the problem of detecting unauthorized data use in ML systems. In Chapter 5, it proposes a general framework that enables a data owner to audit whether her data has been used in training a model, at both the dataset and instance levels. Our auditing method builds upon arbitrary membership-inference techniques and integrates them into an anytime-valid statistical test that we design, which enables the data owner to continuously accumulate data-use evidence through querying the model and adaptively stop at any time while maintaining a quantifiable and provably bounded false-detection rate. Through extensive evaluations on diverse ML models—including image classifiers, visual encoders, LLMs, as well as CLIP and BLIP models, we demonstrate the effectiveness and generality of our approach across a wide range of ML tasks and settings.

Finally, this dissertation investigates the limitations of data-use auditing techniques by proposing AcidWash, a general framework for purifying training data (Chapter 6). We show that by introducing carefully crafted perturbations to potentially marked training inputs, it is possible to mitigate data-use auditing methods while preserving the accuracy of the trained model. Given a reference dataset, identifying such perturbations can be formulated as a min-max optimization problem that can be approximately solved using a two-step procedure. AcidWash performs effectively when the detector used to identify reference data provides high precision, even if it is not perfect. We demonstrate the effectiveness and generality of AcidWash by applying it to two representative data-use auditing methods:

Radioactive Data and our auditing framework proposed in Chapter 5.

Together, these contributions establish a principled foundation for anytime-valid detection of unauthorized data use and clarify the capabilities of statistical auditing in security-critical systems.

7.2 Future Work

An important direction for future research is how to generalize the core idea of this dissertation—developing anytime-valid detection mechanisms—to a broader class of security-critical systems.

Many security problems can be naturally abstracted as detection or binary classification problems. For example, intrusion detection systems aim to determine whether observed network traffic is benign or malicious, fraud detection systems seek to identify whether a transaction is legitimate or fraudulent, and malware detection mechanisms classify software as either benign or malicious. Similarly, credential-breach detection and data-use auditing studied in this dissertation attempt to determine whether sensitive data has been used legitimately or misused by an adversary. These examples suggest that detection mechanisms play a central role across many security domains.

Many security monitoring systems operate continuously and analyze streams of events over time, such as login attempts, network traffic, or model queries. Traditional statistical tests are typically designed for a fixed sample size and may produce misleading results when repeatedly applied during ongoing monitoring. Anytime-valid statistical tests address this limitation by ensuring that the false-detection guarantee holds regardless of when the test is stopped or how often it is evaluated. This property is particularly important for security applications, where detection decisions must be made dynamically as new observations arrive. Developing anytime-valid detection frameworks for a broader class of security-critical systems that remain effective in adversarial environments therefore represents an important and promising direction for future research.

Bibliography

[1] *AB-2013 Generative artificial intelligence: training data transparency*. https://leginfo.legislature.ca.gov/faces/billNavClient.xhtml?bill_id=202320240AB2013. Sept. 2024 (cit. on p. 5).

[2] *AB-375 Privacy: personal information: businesses*. https://leginfo.legislature.ca.gov/faces/billTextClient.xhtml?bill_id=201720180AB375. June 2018 (cit. on p. 5).

[3] M. Abadi, A. Chu, I. Goodfellow, H B. McMahan, I. Mironov, K. Talwar, and L. Zhang. "Deep learning with differential privacy". In: *23rd ACM Conference on Computer and Communications Security*. 2016 (cit. on pp. 130, 183, 197).

[4] Akshima, D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya. "Generation of secure and reliable honeywords, preventing false detection". In: *IEEE Transactions on Dependable and Secure Computing* 16.5 (2019), pp. 757–769 (cit. on pp. 4, 13, 14, 61).

[5] M. H. Almeshekah, C. N Gutierrez, M. J. Atallah, and E. H. Spafford. "ErsatzPasswords: Ending password cracking and detecting password leakage". In: *31st Annual Computer Security Applications Conference*. Dec. 2015, pp. 311–320 (cit. on pp. 12, 13, 97).

[6] S. Alroomi and F. Li. "Measuring website password creation policies at scale". In: *30th ACM Conference on Computer and Communications Security*. 2023 (cit. on pp. 40, 53, 54).

[7] S. Amari. "Backpropagation and stochastic gradient descent method". In: *Neurocomputing* (1993), pp. 185–196 (cit. on p. 121).

[8] S. Baluja. "Hiding images in plain sight: Deep steganography". In: *30th Advances in Neural Information Processing Systems*. 2017 (cit. on p. 17).

[9] M. Basseville, I. V Nikiforov, et al. *Detection of abrupt changes: Theory and application*. 1993 (cit. on p. 72).

[10] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. "Enriching word vectors with subword information". In: *Transactions of the Association for Computational Linguistics* 5 (2017), pp. 135–146 (cit. on p. 28).

[11] L. Bourtoule, V. Chandrasekaran, C. A Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot. "Machine unlearning". In: *42nd IEEE Symposium on Security and Privacy*. 2021 (cit. on p. 157).

[12] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". In: *33th Advances in Neural Information Processing Systems*. 2020 (cit. on pp. 30, 98).

[13] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. "Language models are few-shot learners". In: *33rd Advances in Neural Information Processing Systems*. 2020 (cit. on p. 107).

[14] C. C. S. Caiado and P. N. Rathie. "Polynomial coefficients and distribution of the sum of discrete uniform variables". In: *Annual Conference of the Society of Special Functions and Their Applications*. 2007 (cit. on pp. 115, 118).

[15] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer. "Membership inference attacks from first principles". In: *43rd IEEE Symposium on Security and Privacy*. 2022 (cit. on pp. 5, 6, 17, 143, 144, 146, 147, 150, 167, 170).

[16] N. Carlini, C. Liu, Ú. Erlingsson, J. Kos, and D. Song. "The secret sharer: Evaluating and testing unintended memorization in neural networks". In: *28th USENIX Security Symposium*. 2019 (cit. on pp. 16, 151).

[17] N. Carlini, F. Tramer, E. Wallace, M. Jagielski, A. Herbert-Voss, K. Lee, A. Roberts, T. Brown, D. Song, U. Erlingsson, et al. "Extracting training data from large language models". In: *30th USENIX Security Symposium*. 2021 (cit. on pp. 137, 157).

[18] J. Casal. *1.4 Billion Clear Text Credentials Discovered in a Single Database*. `https://medium.com/4iqdelvedeep/1-4-billion-clear-text-credentials-discovered-in-a-single-database-3131d0a1ae14`. 2017 (cit. on pp. 33, 87).

[19] F. Cayre, C. Fontaine, and T. Furon. "Watermarking security: theory and practice". In: *IEEE Transactions on Signal Processing* 53 (2005), pp. 3976–3987 (cit. on p. 17).

[20] N. Chakraborty, J. Li, V. CM Leung, S. Mondal, Y. Pan, C. Luo, and M. Mukherjee. "Honeyword-based authentication rechniques for protecting passwords: A survey". In: *ACM Computing Surveys* 55 (2022), pp. 1–37 (cit. on p. 65).

[21] V. Chandrasekaran, C. Gao, B. Tang, K. Fawaz, S. Jha, and S. Banerjee. "Face-off: Adversarial face obfuscation". In: *Proceedings on Privacy Enhancing Technologies*. 2021 (cit. on p. 193).

[22] M. Chen, Z. Zhang, T. Wang, M. Backes, and Y. Zhang. "FACE-AUDITOR: Data auditing in facial recognition systems". In: *32nd USENIX Security Symposium*. 2023 (cit. on p. 17).

[23] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. "A simple framework for contrastive learning of visual representations". In: *37th International Conference on Machine Learning*. 2020 (cit. on pp. 32, 33, 90, 107, 133, 134, 151).

[24] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. "A simple framework for contrastive learning of visual representations". In: *37th International Conference on Machine Learning*. 2020 (cit. on p. 107).

[25] Z. Chen and K. Pattabiraman. "Anonymity unveiled: A practical framework for auditing data use in deep learning models". In: *32nd ACM Conference on Computer and Communications Security*. 2025 (cit. on pp. 5, 16).

[26] M. Cherti, R. Beaumont, R. Wightman, M. Wortsman, G. Ilharco, C. Gordon, C. Schuhmann, L. Schmidt, and J. Jitsev. "Reproducible scaling laws for contrastive language-image learning". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2023 (cit. on p. 196).

[27] C. A Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot. "Label-only membership inference attacks". In: *38th International Conference on Machine Learning*. 2021 (cit. on pp. 17, 121, 131).

[28] Ş. Cobzaş, R. Miculescu, and A. Nicolae. *Lipschitz functions*. Vol. 2241. Lecture Notes in Mathematics. Springer, 2019 (cit. on p. 190).

[29] I. N Cofone. *The right to be forgotten: A Canadian and comparative perspective*. Routledge, 2020 (cit. on p. 5).

[30] I. Cohen, Y. Huang, J. Chen, and J. Benesty. "Pearson correlation coefficient". In: *Noise Reduction in Speech Processing* (2009) (cit. on p. 150).

[31] CyberNews. *16 billion passwords exposed in record-breaking data breach: what does it mean for you?* `https://cybernews.com/security/billions-credentials-exposed-infostealers-data-leak/`. 2025 (cit. on pp. 2, 4).

[32] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang. "The tangled web of password reuse". In: *ISOC Network and Distributed System Security Symposium*. 2014 (cit. on p. 34).

[33] A. De Brebisson and P. Vincent. "An exploration of softmax alternatives belonging to the spherical loss family". In: *4th International Conference for Learning Representations*. 2016 (cit. on p. 144).

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li. "Imagenet: A large-scale hierarchical image database". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2009 (cit. on pp. 107, 113, 119, 120, 140, 186).

[35] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer. "Qlora: Efficient finetuning of quantized LLMs". In: *36th Advances in Neural Information Processing Systems*. 2024 (cit. on p. 137).

[36] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: (2019) (cit. on pp. 107, 136).

[37] A. Dionysiou and E. Athanasopoulos. "Lethe: Practical data breach detection with zero persistent secret state". In: *7th IEEE European Symposium on Security and Privacy*. June 2022 (cit. on pp. 12, 13, 97).

[38] A. Dionysiou, V. Vassiliades, and E. Athanasopoulos. "Honeygen: Generating honeywords using representation learning". In: *16thACM Symposium on Information, Computer and Communications Security*. 2021 (cit. on pp. 13, 25, 28, 64).

[39] L. Du, M. Chen, M. Sun, S. Ji, P. Cheng, J. Chen, and Z. Zhang. "ORL-AUDITOR: Dataset auditing in offline deep reinforcement learning". In: *31st ISOC Network and Distributed System Security Symposium*. 2024 (cit. on pp. 5, 17).

[40] L. Du, X. Zhou, M. Chen, C. Zhang, Z. Su, P. Cheng, J. Chen, and Z. Zhang. "SoK: Dataset copyright auditing in machine learning systems". In: *46st IEEE Symposium on Security and Privacy*. 2025 (cit. on p. 15).

[41] M. Dürmuth, F. Angelstorf, C. Castelluccia, D. Perito, and A. Chaabane. "OMEN: Faster password guessing using an ordered markov enumerator". In: *7th International Symposium on Engineering Secure Software and Systems*. 2015 (cit. on p. 88).

[42] C. Dwork. "Differential privacy". In: *33rd International Colloquium on Automata, Languages, and Programming*. 2006 (cit. on p. 18).

[43] A. Dziedzic, H. Duan, M. A. Kaleem, N. Dhawan, J. Guan, Y. Cattan, F. Boenisch, and N. Papernot. "Dataset inference for self-supervised models". In: *Advances in Neural Information Processing Systems*. 2022 (cit. on p. 5).

[44] I. Erguler. "Achieving flatness: Selecting the honeywords from existing user passwords". In: *IEEE Transactions on Parallel and Distributed Systems* 13.2 (2016) (cit. on pp. 4, 13, 14, 61).

[45] P. Fernandez, G. Couairon, H. Jégou, M. Douze, and T. Furon. "The stable signature: Rooting watermarks in latent diffusion models". In: *IEEE International Conference on Computer Vision*. 2023 (cit. on p. 17).

[46] D. Florêncio, C. Herley, and P. C. van Oorschot. "An administrator's guide to internet password research". In: *28th Large Installation System Administration Conference*. Nov. 2014, pp. 35–52 (cit. on pp. 36, 46, 58, 84).

[47] K. Gao, Y. Bai, J. Gu, Y. Yang, and S.-T. Xia. "Backdoor defense via adaptively splitting poisoned dataset". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2023 (cit. on pp. 183, 187, 198).

[48] J. Geiping, L. Fowl, W R. Huang, W. Czaja, G. Taylor, M. Moeller, and T. Goldstein. "Witches' brew: Industrial scale data poisoning via gradient matching". In: *9th International Conference for Learning Representations*. 2021 (cit. on pp. 18, 19, 198).

[49] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg. "Badnets: Evaluating backdooring attacks on deep neural networks". In: *IEEE Access* 7 (2019), pp. 47230–47244 (cit. on p. 16).

[50] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C Courville. "Improved training of wasserstein GANs". In: *31th Advances in Neural Information Processing Systems*. 2017 (cit. on pp. 10, 182, 190, 193, 196).

[51] J. Guo, Y. Li, L. Wang, S.-T. Xia, H. Huang, C. Liu, and B. Li. "Domain watermark: Effective and harmless dataset copyright protection is closed at hand". In: *38th Advances in Neural Information Processing Systems*. 2024 (cit. on pp. 5, 6, 15).

[52] Y. Guo, Z. Zhang, and Y. Guo. "Superword: A honeyword system for achieving higher security goals". In: *Computers & Security* (2021) (cit. on pp. 4, 13, 61).

[53] R. A Haddad, A. N Akansu, et al. "A class of fast Gaussian binomial filters for speech and image processing". In: *IEEE Transactions on Signal Processing* (1991) (cit. on p. 147).

[54] K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016 (cit. on pp. 107, 119, 120, 129, 142, 149).

[55] Y. He, B. Li, Y. Wang, M. Yang, J. Wang, H. Hu, and X. Zhao. "Is difficulty calibration all we need? Towards more practical membership inference attacks". In: *31th ACM Conference on Computer and Communications Security*. 2024 (cit. on p. 6).

[56] K. Hill. *The secretive company that might end privacy as we know it*. `https://www.nytimes.com/2020/01/18/technology/clearview-privacy-facial-recognition.html`. 181 2020 (cit. on p. 5).

[57] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural Computation* 9.8 (1997) (cit. on p. 27).

[58] S. Hong, V. Chandrasekaran, Y. Kaya, T. Dumitraş, and N. Papernot. "On the effectiveness of mitigating data poisoning attacks with gradient shaping". In: *arXiv preprint arXiv:2002.11497* (2020) (cit. on pp. 18, 130).

[59] H. Hu, Z. Salcic, L. Sun, G. Dobbie, P. S Yu, and X. Zhang. "Membership inference attacks on machine learning: A survey". In: *ACM Computing Surveys* 54 (2022), pp. 1–37 (cit. on p. 17).

[60] T. Huang, G. Yang, and G. Tang. "A fast two-dimensional median filtering algorithm". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* (1979) (cit. on p. 147).

[61] Z. Huang, L. Bauer, and M. K. Reiter. "The impact of exposed passwords on honeyword efficacy". In: *33rd USENIX Security Symposium*. 2024 (cit. on pp. 14, 22, 64, 66, 84, 87, 88, 90).

[62] Z. Huang, N. Z. Gong, and M. K. Reiter. "A general framework for data-use auditing of ML models". In: *31st ACM Conference on Computer and Communications Security*. 2024 (cit. on pp. 5, 18, 108, 152, 154, 185, 186).

[63] IBM Security. *Cost of a Data Breach Report 2025*. `https://www.ibm.com/reports/data-breach`. 2025 (cit. on pp. 3, 90).

[64] G. Ilharco, M. Wortsman, R. Wightman, C. Gordon, N. Carlini, R. Taori, A. Dave, V. Shankar, H. Namkoong, J. Miller, H. Hajishirzi, A. Farhadi, and L. Schmidt. *OpenCLIP*. Version 0.1. July 2021. DOI: `10.5281/zenodo.5143773`. URL: `https://doi.org/10.5281/zenodo.5143773` (cit. on p. 196).

[65] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *3rd International Conference for Learning Representations*. 2015 (cit. on pp. 128, 149).

[66] A. Iscen, T. Furon, V. Gripon, M. Rabbat, and H. Jégou. "Memory vectors for similarity search in high-dimensional spaces". In: *IEEE Transactions on Big Data* 4 (2017), pp. 65–77 (cit. on p. 123).

[67] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong. "Memguard: Defending against black-box membership inference attacks via adversarial examples". In: *26th ACM Conference on Computer and Communications Security*. 2019 (cit. on p. 130).

[68] N. L Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions*. 2nd ed. Wiley, 1995 (cit. on p. 146).

[69] A. Juels and R. L. Rivest. "Honeywords: Making password-cracking detectable". In: *20th ACM Conference on Computer and Communications Security*. Nov. 2013 (cit. on pp. 3, 12–14, 25, 28, 51, 52, 63, 64, 68, 88).

[70] D. P Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference for Learning Representations*. 2015 (cit. on pp. 27, 42, 139, 192).

[71] M. Ko, M. Jin, C. Wang, and R. Jia. "Practical membership inference attacks against large-scale multi-modal models: A pilot study". In: *IEEE International Conference on Computer Vision*. 2023 (cit. on pp. 5, 139, 153, 157).

[72] A. Krizhevsky. "Learning multiple layers of features from tiny images". MA thesis. University of Toronto, 2009 (cit. on pp. 119, 140, 151, 194).

[73] A. Krizhevsky, I. Sutskever, and G. E Hinton. "Imagenet classification with deep convolutional neural networks". In: *25th Advances in Neural Information Processing Systems*. 2012 (cit. on p. 128).

[74] R. J Larsen and M. L Marx. *An introduction to mathematical statistics*. Prentice Hall Hoboken, NJ, 2005 (cit. on p. 124).

[75] Y. Le and X. S. Yang. *Tiny ImageNet Visual Recognition Challenge*. `http://vision.stanford.edu/teaching/cs231n/reports/2015/pdfs/yle_project.pdf`. 2015 (cit. on pp. 119, 140, 151, 194).

[76] M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. "Certified robustness to adversarial examples with differential privacy". In: *40th IEEE Symposium on Security and Privacy*. 2019 (cit. on p. 18).

[77] V. I Levenshtein et al. "Binary codes capable of correcting deletions, insertions, and reversals". In: *Soviet Physics Doklady*. Vol. 10. Soviet Union. 1966, pp. 707–710 (cit. on p. 136).

[78] B. Li, Y. Wei, Y. Fu, Z. Wang, Y. Li, J. Zhang, R. Wang, and T. Zhang. "Towards reliable verification of unauthorized data usage in personalized text-to-image diffusion models". In: *46th IEEE Symposium on Security and Privacy*. 2024 (cit. on p. 16).

[79] J. Li, D. Li, C. Xiong, and S. Hoi. "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation". In: *39th International Conference on Machine Learning*. 2022 (cit. on p. 107).

[80] Y. Li, Y. Bai, Y. Jiang, Y. Yang, S.-T. Xia, and B. Li. "Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection". In: *36th Advances in Neural Information Processing Systems*. 2022 (cit. on pp. 5, 6, 15, 16, 107, 110, 122, 125).

[81] Y. Li, M. Zhu, X. Yang, Y. Jiang, T. Wei, and S.-T. Xia. "Black-box dataset ownership verification via backdoor watermarking". In: *IEEE Transactions on Information Forensics and Security* 18 (2023), pp. 2318–2332 (cit. on pp. 5, 6, 15, 16).

[82] C.-J. Lin. "Projected gradient methods for nonnegative matrix factorization". In: *Neural Computation* (2007) (cit. on p. 120).

[83] H. Liu, J. Jia, W. Qu, and N. Z. Gong. "EncoderMI: Membership inference against pre-trained encoders in contrastive learning". In: *28th ACM Conference on Computer and Communications Security*. 2021 (cit. on pp. 5, 17, 134, 152, 157).

[84] T. Y. Liu, Y. Yang, and B. Mirzasoleiman. "Friendly noise against adversarial noise: a powerful defense against data poisoning attack". In: *36th Advances in Neural Information Processing Systems*. 2022 (cit. on pp. 19, 183, 197, 198).

[85] Y. Long, L. Wang, D. Bu, V. Bindschaedler, X. Wang, H. Tang, C. A Gunter, and K. Chen. "A pragmatic approach to membership inferences on machine learning models". In: *5th IEEE European Symposium on Security and Privacy*. 2020 (cit. on p. 17).

[86] I. Loshchilov and F. Hutter. "Sgdr: Stochastic gradient descent with warm restarts". In: *5th International Conference for Learning Representations*. 2017 (cit. on p. 134).

[87] I. Loshchilov and F. Hutter. "Fixing weight decay regularization in adam". In: *6th International Conference for Learning Representations*. 2018 (cit. on p. 137).

[88] I. Loshchilov and F. Hutter. "Decoupled weight decay regularization". In: *7th International Conference for Learning Representations*. 2019 (cit. on p. 153).

[89] X. Luo, R. Zhan, H. Chang, F. Yang, and P. Milanfar. "Distortion agnostic deep watermarking". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cit. on p. 17).

[90] J. Ma, W. Yang, M. Luo, and N. Li. "A study of probabilistic password models". In: *35th IEEE Symposium on Security and Privacy*. 2014 (cit. on pp. 13, 15, 25, 64, 88).

[91] Y. Ma, X. Zhu, and J. Hsu. "Data poisoning against differentially-private learners: Attacks and defenses". In: *33rd International Joint Conference on Artificial Intelligence*. 2019 (cit. on p. 18).

[92] Infosecurity Magazine. *FBI Warns of $262M losses from account takeover fraud in 2025*. `https://www.infosecurity-magazine.com/news/fbi-warns-account-takeover-fraud/`. 2025 (cit. on p. 3).

[93] P. Maini, M. Yaghini, and N. Papernot. "Dataset inference: Ownership resolution in machine learning". In: *9th International Conference for Learning Representations*. 2021 (cit. on p. 5).

[94] A. Mantelero. "The EU proposal for a general data protection regulation and the roots of the 'right to be forgotten'". In: *Computer Law & Security Review* (2013) (cit. on p. 5).

[95] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv. "Why users (don't) use password managers at a large educational institution". In: *31st USENIX Security Symposium*. Aug. 2022 (cit. on p. 59).

[96] W. Melicher, B. Ur, S. M. Segreti, S. Komanduri, L. Bauer, N. Christin, and L. F. Cranor. "Fast, lean, and accurate: Modeling password guessability using neural networks". In: *25th USENIX Security Symposium*. 2016 (cit. on pp. 25–27, 88).

[97] Y. Miao, M. Xue, C. Chen, L. Pan, J. Zhang, B. Z. H. Zhao, D. Kaafar, and Y. Xiang. "The audio auditor: user-level membership inference in internet of things voice services". In: *21st Privacy Enhancing Technologies Symposium*. 2021 (cit. on p. 17).

[98] Microsoft. *Store passwords using reversible encryption*. `https://learn.microsoft.com/en-us/windows/security/threat-protection/security-policy-settings/store-passwords-using-reversible-encryption`. Dec. 2022 (cit. on p. 40).

[99] M. Mitzenmacher and E. Upfal. *Probability and computing: Randomization and probabilistic techniques in algorithms and data analysis*. Cambridge University Press, 2005 (cit. on p. 83).

[100] S. Mohammad, F. Bravo-Marquez, M. Salameh, and S. Kiritchenko. "Semeval-2018 task 1: Affect in tweets". In: *International Workshop on Semantic Evaluation*. 2018 (cit. on p. 136).

[101] A. Mukherjee, K. Murali, S. K. Jha, N. Ganguly, R. Chatterjee, and M. Mondal. "MASCARA: Systematically generating memorable and secure passphrases". In: *18th ACM Symposium on Information, Computer and Communications Security*. 2023 (cit. on p. 23).

[102] M. Nasr, R. Shokri, and A. Houmansadr. "Machine learning with membership privacy using adversarial regularization". In: *25th ACM Conference on Computer and Communications Security*. 2018 (cit. on p. 131).

[103] W. Nie, B. Guo, Y. Huang, C. Xiao, A. Vahdat, and A. Anandkumar. "Diffusion models for adversarial purification". In: *39th International Conference on Machine Learning*. 2022 (cit. on p. 19).

[104] OpenAI. *ChatGPT*. `https://chat.openai.com/`. 2025 (cit. on p. 98).

[105] E. S Page. "Continuous inspection schemes". In: *Biometrika* (1954) (cit. on p. 72).

[106] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. "Beyond credential stuffing: Password similarity models using neural networks". In: *40th IEEE Security and Privacy*. May 2019 (cit. on pp. 29, 34, 87, 88).

[107] X. Pan, M. Zhang, S. Ji, and M. Yang. "Privacy risks of general-purpose language models". In: *41st IEEE Symposium on Security and Privacy*. 2020 (cit. on pp. 5, 17, 156, 157).

[108] D. Pasquini, A. Gangwal, G. Ateniese, M. Bernaschi, and M. Conti. "Improving password guessing via representation learning". In: *42nd IEEE Symposium on Security and Privacy*. 2021 (cit. on p. 13).

[109] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *33rd Advances in Neural Information Processing Systems*. 2019 (cit. on p. 144).

[110] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget. "Let's go in for a closer look: Observing passwords in their natural habitat". In: *24th ACM Conference on Computer and Communications Security*. Oct. 2017 (cit. on pp. 3, 4, 35).

[111] N. Peri, N. Gupta, W R. Huang, L. Fowl, C. Zhu, S. Feizi, T. Goldstein, and J. P Dickerson. "Deep k-NN defense against clean-label data poisoning attacks". In: *16th European Conference on Computer Vision Workshop*. 2020 (cit. on pp. 18, 195).

[112] J. Peters, D. Janzing, and B. Schölkopf. *Elements of causal inference: foundations and learning algorithms*. The MIT Press, 2017 (cit. on p. 198).

[113] V. L. Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczyński, and W. Joosen. "Tranco: A research-oriented top sites ranking hardened against manipulation". In: *25th ISOC Network and Distributed System Security Symposium*. Feb. 2018 (cit. on pp. 40, 53).

[114] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. A., G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. "Learning transferable visual models from natural language supervision". In: *38th International Conference on Machine Learning*. 2021 (cit. on pp. 107, 129, 138, 139, 186, 196).

[115] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. "Zero-shot text-to-image generation". In: *38th International Conference on Machine Learning*. 2021 (cit. on p. 138).

[116] Dark Reading. *Study finds 15 billion stolen, exposed credentials in criminal markets*. `https://www.darkreading.com/cyberattacks-data-breaches/study-finds-15-billion-stolen-exposed-credentials-in-criminal-markets`. 2020 (cit. on p. 98).

[117] E. Rosenfeld, E. Winston, P. Ravikumar, and Z. Kolter. "Certified robustness to label-flipping attacks via randomized smoothing". In: *37th International Conference on Machine Learning*. 2020 (cit. on p. 18).

[118] A. Sablayrolles, M. Douze, C. Schmid, and H. Jégou. "Radioactive data: Tracing through training". In: *37th International Conference on Machine Learning*. 2020 (cit. on pp. 5, 6, 15, 16, 107, 110, 122, 183–186, 193, 199, 211).

[119] A. Sablayrolles, M. Douze, C. Schmid, Y. Ollivier, and H. Jégou. "White-box vs black-box: Bayes optimal strategies for membership inference". In: *36th International Conference on Machine Learning*. 2019 (cit. on p. 17).

[120] A. Saha, A. Subramanya, and H. Pirsiavash. "Hidden trigger backdoor attacks". In: *34th International Joint Conference on Artificial Intelligence*. 2020 (cit. on p. 16).

[121] A. Salem, Y. Zhang, M. Humbert, P. Berrang, M. Fritz, and M. Backes. "Ml-leaks: Model and data independent membership inference attacks and defenses on machine learning models". In: *26th ISOC Network and Distributed System Security Symposium*. 2019 (cit. on pp. 5, 17).

[122] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. "Mobilenetv2: Inverted residuals and linear bottlenecks". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2018 (cit. on p. 128).

[123] C. Schuhmann, R. Beaumont, R. Vencu, C. W Gordon, R. Wightman, M. Cherti, T. Coombes, A. Katta, C. Mullis, M. Wortsman, P. Schramowski, S. R Kundurthy, K. Crowson, L. Schmidt, R. Kaczmarczyk, and J. Jitsev. "LAION-5B: An open large-scale dataset for training next generation image-text models". In: *36th Advances in Neural Information Processing Systems*. 2022. URL: https://openreview.net/forum?id=M3Y74vmsMcY (cit. on p. 196).

[124] A. Shafahi, W R. Huang, M. Najibi, O. Suciu, C. Studer, T. Dumitras, and T. Goldstein. "Poison frogs! Targeted clean-label poisoning attacks on neural networks". In: *32st Advances in Neural Information Processing Systems*. 2018 (cit. on p. 18).

[125] Shape Security. *2018 Credential Spill Report*. https://info.shapesecurity.com/rs/935-ZAM-778/images/Shape_Credential_Spill_Report_2018.pdf. 2018 (cit. on p. 3).

[126] W. Shi, A. Ajith, M. Xia, Y. Huang, D. Liu, T. Blevins, D. Chen, and L. Zettlemoyer. "Detecting pretraining data from large language models". In: *12th International Conference for Learning Representations*. 2024 (cit. on pp. 5, 157).

[127] J. Shin, A. Ramdas, and A. Rinaldo. "E-detectors: A nonparametric framework for sequential change detection". In: *New England Journal of Statistics in Data Science* (2024) (cit. on p. 72).

[128] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. "Membership inference attacks against machine learning models". In: *38th IEEE Symposium on Security and Privacy*. 2017 (cit. on pp. 5, 6, 17).

[129] Similarweb. *Top Websites Ranking*. `https://www.similarweb.com/top-websites/`. 2023 (cit. on pp. 53, 54).

[130] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *3rd International Conference for Learning Representations*. 2015 (cit. on pp. 107, 119, 128, 129, 149).

[131] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Ng, and C. Potts. "Recursive deep models for semantic compositionality over a sentiment treebank". In: *ACL Conference on Empirical Methods in Natural Language Processing*. 2013 (cit. on p. 136).

[132] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep unsupervised learning using nonequilibrium thermodynamics". In: *32nd International Conference on Machine Learning*. 2015 (cit. on p. 19).

[133] D. M. Sommer, L. Song, S. Wagh, and P. Mittal. "Towards probabilistic verification of machine unlearning". In: *Proceedings on Privacy Enhancing Technologies*. 2022 (cit. on p. 157).

[134] C. Song, T. Ristenpart, and V. Shmatikov. "Machine learning models that remember too much". In: *24th ACM Conference on Computer and Communications Security*. 2017 (cit. on pp. 9, 106, 115).

[135] C. Song and V. Shmatikov. "Auditing data provenance in text-generation models". In: *25th ACM International Conference on Knowledge Discovery & Data Mining*. 2019 (cit. on p. 17).

[136] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese. "Deep metric learning via lifted structured feature embedding". In: *29th IEEE Conference on Computer Vision and Pattern Recognition*. 2016 (cit. on p. 32).

[137] L. Song and P. Mittal. "Systematic evaluation of privacy risks of machine learning models". In: *30th USENIX Security Symposium*. 2021 (cit. on pp. 17, 114).

[138] H. Souri, L. Fowl, R. Chellappa, M. Goldblum, and T. Goldstein. "Sleeper agent: Scalable hidden trigger backdoors for neural networks trained from scratch". In: *35thAdvances in Neural Information Processing Systems*. 2022 (cit. on p. 198).

[139] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning". In: *30th International Conference on Machine Learning*. 2013 (cit. on p. 134).

[140] I. Sutskever, J. Martens, and G. E. Hinton. "Generating text with recurrent neural networks". In: *28thInternational Conference on Machine Learning*. 2011 (cit. on p. 26).

[141] M. Tancik, B. Mildenhall, and R. Ng. "Stegastamp: Invisible hyperlinks in physical photographs". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2020 (cit. on p. 17).

[142] R. Tang, Q. Feng, N. Liu, F. Yang, and X. Hu. "Did you train on my dataset? Towards public dataset protection with cleanlabel backdoor watermarking". In: *ACM SIGKDD Explorations Newsletter* (2023) (cit. on pp. 5, 6, 15, 16).

[143] The New York Times. *The Times Sues OpenAI and Microsoft Over A.I. Use of Copyrighted Work*. `https://www.nytimes.com/2023/12/27/business/media/new-york-times-open-ai-microsoft-lawsuit.html`. 2023 (cit. on p. 5).

[144] K. Thomas, F. Li, A. Zand, J. Barrett, J. Ranieri, L. Invernizzi, Y. Markov, O. Comanescu, V. Eranti, A. Moscicki, D. Margolis, V. Paxson, and E. Bursztein. "Data breaches, phishing, or malware? Understanding the risks of stolen credentials". In: *24th ACM Conference on Computer and Communications Security*. 2017 (cit. on pp. 3, 98).

[145] K. Toubba. *Security incident update and recommended actions*. `https://blog.lastpass.com/2023/03/security-incident-update-recommended-actions/`. Jan. 2023 (cit. on p. 59).

[146] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al. "Llama 2: Open foundation and fine-tuned chat models". In: *arXiv* 2307.09288 (2023) (cit. on pp. 107, 135, 137).

[147] B. Ur, S. M. Segreti, L. Bauer, N. Christin, L. F. Cranor, S. Komanduri, D. Kurilova, M. L. Mazurek, W. Melicher, and R. Shay. "Measuring real-world accuracies and biases in modeling password guessability". In: *24th USENIX Security Symposium*. 2015 (cit. on p. 23).

[148] P. C. van Oorschot. *Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin*. 2nd. Springer, 2021 (cit. on p. 23).

[149] L. N. Vaserstein. "Markov processes over denumerable products of spaces, describing large systems of automata". In: *Problemy Peredachi Informatsii* 5 (1969), pp. 64–72 (cit. on pp. 10, 182).

[150] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *30th Advances in Neural Information Processing Systems*. 2017 (cit. on pp. 98, 135).

[151] A. Vigderman. *Password Manager Annual Report 2022*. `https://www.security.org/digital-safety/password-manager-annual-report/`. 27 1 2023 (cit. on p. 43).

[152] C. Villani. *Optimal transport: old and new*. 2009 (cit. on pp. 183, 190).

[153] V. Vorobev and M. Kuznetsov. *A paraphrasing model based on ChatGPT paraphrases*. `https://huggingface.co/humarin/chatgpt_paraphraser_on_T5_base`. 2023 (cit. on p. 136).

[154] B. Wang, Y. Yao, S. Shan, H. Li, B. Viswanath, H. Zheng, and B. Y Zhao. "Neural cleanse: Identifying and mitigating backdoor attacks in neural networks". In: *40th IEEE Symposium on Security and Privacy*. 2019 (cit. on p. 187).

[155] D. Wang, H. Cheng, P. Wang, X. Huang, and G. Jian. "Zipf's law in passwords". In: *IEEE Transactions on Information Forensics and Security*. Vol. 12. 11. Nov. 2017, pp. 2776–2791 (cit. on p. 28).

[156] D. Wang, H. Cheng, P. Wang, J. Yan, and X. Huang. "A security analysis of honeywords". In: *25th ISOC Network and Distributed System Security Symposium*. Feb. 2018 (cit. on pp. 13–15, 22, 25, 66, 84, 85).

[157] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang. "How to attack and generate honeywords". In: *43rd IEEE Symposium on Security and Privacy*. May 2022 (cit. on pp. 4, 13–15, 25, 29, 39, 51, 61, 64–66, 68, 84, 88, 90).

[158] D. Wang, Y. Zou, Y. Xiao, S. Ma, and X. Chen. "PASS2EDIT: A multi-Step generative model for guessing edited passwords". In: *32nd USENIX Security Symposium*. 2023 (cit. on p. 88).

[159] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin. "Deep metric learning with angular loss". In: *31st IEEE International Conference on Computer Vision*. 2017 (cit. on p. 32).

[160] K. C. Wang and M. K Reiter. "Bernoulli honeywords". In: *31st ISOC Network and Distributed System Security Symposium*. 2024 (cit. on pp. 4, 13, 14, 53, 61, 69, 88).

[161] K. C. Wang and M. K. Reiter. "How to end password reuse on the web". In: *26th ISOC Network and Distributed System Security Symposium*. Feb. 2019 (cit. on p. 59).

[162] K. C. Wang and M. K. Reiter. "Using Amnesia to detect credential database breaches". In: *30th USENIX Security Symposium*. Aug. 2021 (cit. on pp. 12, 13, 97).

[163] Z. Wang, C. Chen, L. Lyu, D. N Metaxas, and S. Ma. "DIAGNOSIS: Detecting unauthorized data usages in text-to-image diffusion models". In: *12th International Conference for Learning Representations*. 2024 (cit. on pp. 5, 15, 16).

[164] R. Wash, E. Rader, R. Berman, and Z. Wellmer. "Understanding password choices: How frequently entered passwords are re-used across websites". In: *Symposium on Usable Privacy and Security*. 2016 (cit. on pp. 3, 4).

[165] I. Waudby-Smith and A. Ramdas. "Confidence sequences for sampling without replacement". In: *33rd Advances in Neural Information Processing Systems*. 2020 (cit. on pp. 116, 118).

[166] J. T.-Z. Wei, R. Y. Wang, and R. Jia. "Proving membership in LLM pretraining data via data watermarks". In: *Findings of the Association for Computational Linguistics*. 2024 (cit. on pp. 15, 16).

[167] M. Weir, S. Aggarwal, B. De Medeiros, and B. Glodek. "Password cracking using probabilistic context-free grammars". In: *30th IEEE Symposium on Security and Privacy*. 2009 (cit. on pp. 13, 15, 25, 64, 88).

[168] E. Wenger, X. Li, B. Y Zhao, and V. Shmatikov. "Data isotopes for data provenance in DNNs". In: *Privacy Enhancing Technologies Symposium*. 2024 (cit. on pp. 5, 6, 15, 110).

[169] M. Xu, C. Wang, J. Yu, J. Zhang, K. Zhang, and W. Han. "Chunk-level password guessing: Towards modeling refined password composition representations". In: *28th ACM Conference on Computer and Communications Security*. Nov. 2021 (cit. on pp. 25, 30, 64).

[170] M. Xu, J. Yu, X. Zhang, C. Wang, S. Zhang, H. Wu, and W. Han. "Improving real-world password guessing attacks via bi-directional transformers". In: *32nd USENIX Security Symposium*. 2023 (cit. on pp. 13, 87).

[171] J. Ye, A. Maddi, S. K. Murakonda, V. Bindschaedler, and R. Shokri. "Enhanced membership inference attacks against machine learning models". In: *29th ACM Conference on Computer and Communications Security*. 2022 (cit. on pp. 5, 6, 17, 143, 146, 147, 167).

[172] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha. "Privacy risk in machine learning: Analyzing the connection to overfitting". In: *31st IEEE Computer Security Foundations Symposium*. 2018 (cit. on p. 17).

[173] J. Yoon, S. J. Hwang, and J. Lee. "Adversarial purification with score-based generative models". In: *38th International Conference on Machine Learning*. 2021 (cit. on p. 19).

[174] P. Young, A. Lai, M. Hodosh, and J. Hockenmaier. "From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions". In: *Transactions of the Association for Computational Linguistics* (2014), pp. 67–78 (cit. on p. 138).

[175] F. Yu and M. V. Martin. "Honey, I chunked the passwords: Generating semantic honeywords resistant to targeted attacks using pre-trained language models". In:

International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. 2023 (cit. on pp. 4, 13, 14, 30, 31, 61).

[176] N. Yu, V. Skripniuk, S. Abdelnabi, and M. Fritz. "Artificial fingerprinting for generative models: Rooting deepfake attribution in training data". In: *IEEE International Conference on Computer Vision*. 2021 (cit. on pp. 15, 17).

[177] S. Zagoruyko. "Wide residual networks". In: *arXiv preprint arXiv:1605.07146* (2016) (cit. on p. 149).

[178] S. Zarifzadeh, P. Liu, and R. Shokri. "Low-cost high-power membership inference attacks". In: *41st International Conference on Machine Learning*. 2024 (cit. on pp. 5, 6, 143, 144, 146, 147, 167).

[179] Y. Zeng, S. Chen, W. Park, Z M. Mao, M. Jin, and R. Jia. "Adversarial unlearning of backdoors via implicit hypergradient". In: *9th International Conference for Learning Representations*. 2021 (cit. on p. 187).

[180] Y. Zeng, M. Pan, H. Jahagirdar, M. Jin, L. Lyu, and R. Jia. "Meta-Sift: How to sift out a clean subset in the presence of data poisoning?" In: *32ndUSENIX Security Symposium*. 2023 (cit. on p. 18).

[181] J. Zhang, D. Das, G. Kamath, and F. Tramèr. "Position: Membership inference attacks cannot prove that a model was trained on your data". In: *IEEE Conference on Secure and Trustworthy Machine Learning*. 2025 (cit. on pp. 6, 183, 186).

[182] X. Zhang, J. Zhao, and Y. LeCun. "Character-level convolutional networks for text classification". In: *29th Advances in Neural Information Processing Systems*. 2015 (cit. on p. 136).

[183] Y. Zhang, A. Albarghouthi, and L. D'Antoni. "BagFlip: A certified defense against data poisoning". In: *36th Advances in Neural Information Processing Systems*. 2022 (cit. on p. 18).

[184] Z. Zhang, Q. Liu, Z. Wang, Z. Lu, and Q. Hu. "Backdoor defense via deconfounded representation learning". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2023 (cit. on pp. 183, 198).

[185] J. Zhu, R. Kaplan, J. Johnson, and F.-F. Li. "Hidden: Hiding data with deep networks". In: *European Conference on Computer Vision*. 2018 (cit. on p. 17).

[186] J. Zhu, J. Zha, D. Li, and L. Wang. "A unified membership inference method for visual self-supervised encoder via part-aware capability". In: *31th ACM Conference on Computer and Communications Security*. 2024 (cit. on p. 152).

[187] Z. Zou, B. Gong, and L. Wang. “Anti-neuron watermarking: protecting personal data against unauthorized neural networks”. In: *European Conference on Computer Vision*. 2022 (cit. on p. 15).

Biography

Zonghao Huang graduated with a Ph.D. degree in Computer Science from Duke University in 2026, under the supervision of Professor Michael K. Reiter. His research interests broadly lie in computer security and privacy, with a particular focus on the intersection of computer security, applied statistics, and machine learning. He developed data-auditing schemes for authentication and machine-learning systems to detect unauthorized data access and use. A central objective of his work is to ensure strong statistical guarantees, designing anytime-valid detection mechanisms that continuously accumulate detection evidence and adaptively stop at any time while maintaining tunable, provably bounded false-detection rates. His research has been published on top-tier security conferences, including USENIX Security and ACM CCS. Before Duke, he received his master degree at Oklahoma State University and bachelor degree at Xiamen University.