# Building and Evaluating Privacy-Preserving Data Processing Systems

*Luca Melis*

A dissertation submitted in partial fulfillment

of the requirements for the degree of

**Doctor of Philosophy**

of

**University College London**.

Department of Computer Science

University College London

I, Luca Melis, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

“Non sai che fare, non sai dove andare, miagoli nel buio.”

(Quelo)

# Abstract

Large-scale data processing prompts a number of important challenges, including guaranteeing that collected or published data is not misused, preventing disclosure of sensitive information, and deploying privacy protection frameworks that support usable and scalable services.

In this dissertation, we study and build systems geared for privacy-friendly data processing, enabling computational scenarios and applications where potentially sensitive data can be used to extract useful knowledge, and which would otherwise be impossible without such strong privacy guarantees. For instance, we show how to privately and efficiently aggregate data from many sources and large streams, and how to use the aggregates to extract useful statistics and train simple machine learning models. We also present a novel technique for privately releasing generative machine learning models and entire high-dimensional datasets produced by these models. Finally, we demonstrate that the data used by participants in training generative and collaborative learning models may be vulnerable to inference attacks and discuss possible mitigation strategies.

# Impact Statement

The results presented in this dissertation will likely facilitate the design, development, and evaluation of innovative techniques for privacy-aware data collection and machine learning. Stakeholders and practitioners will thus be able to overcome the increasingly relevant tension between the utility of extracting knowledge from data and the responsibility to protect individuals' privacy. More specifically, this dissertation introduces novel techniques for privacy-preserving collection of statistics, allowing providers to instantiate protocols and build systems addressing "real-world" applications. To this end, we have designed and developed a scalable and easy-to-integrate framework supporting privacy-preserving computation of analytics based on large-scale and private data aggregation. Further, in a first-of-its-kind attempt to build a private generative machine learning model based on neural networks, our research will allow companies to generate and share synthetic high-dimensional data without incurring risks of privacy breaches.

This dissertation also sets out to study how machine learning models may lead to information leakage, thus addressing important academic and policy challenges. Our research identifies a few important gaps in the academic literature related to the evaluation of membership inference in both collaborative learning and generative machine learning models, as well as to the investigation of property inference in collaborative learning. In particular, membership inference can directly violate privacy if inclusion in a training set is itself sensitive. For example, if synthetic health-related images (generated by generative models) are used, e.g., for research purposes, discovering that a specific record

was used for training leaks information about the individual's health. Therefore, regulators can use membership inference to support the suspicion that a model was trained on personal data without an adequate legal basis, or for a purpose not compatible with the data collection, e.g., to detect violations of data protection regulations such as the new EU General Data Protection Regulation (GDPR). Machine learning as a service (MLaaS) providers can use our inference attacks as a benchmark before allowing third parties access to the model; providers may restrict access in case the inference attack yields good results.

Finally, the academic community at large can benefit from our work, in that it advances the state of the art in studying and addressing the challenges of privacy-preserving analytics in innovative and more effective ways, as well as motivating the need for future research on investigating better defenses against inference attacks. Therefore, we are confident our research will promote interdisciplinary collaborations at the intersection of security/privacy and machine learning.

# Acknowledgements

First and foremost I want to thank my advisor Emiliano De Cristofaro. It has been an honor to be his first doctoral student. I am grateful for all his contributions of time and ideas to make my research experience productive and stimulating. The passion he has for research has been extremely motivational for me, even during (countless) tough times in my Ph.D. pursuit.

I would also like to express my gratitude to my viva examiners, Sebastian Riedel and Hamed Haddadi, for their precious feedback on my dissertation.

I have been very lucky to work with great researchers: Jamie Hayes, Apostolos Pyrgelis, Conghzeng Song, Cyril Soldani, George Danezis, Gergely Acs, Gábor Gulyás, Vitaly Shmatikov, Hassan Jameel Asghar, Mohamed Ali Kaafar, and Laurent Mathy.

My sincere thanks also go to Claude Castelluccia and Baris Coskun, who provided me the invaluable opportunity to join their research groups at INRIA and AWS as intern.

I gratefully acknowledge the funding sources that made my research work possible. I was funded by the UCL Computer Science department and was honored to be an enrichment student at The Alan Turing Institute.

And finally, last but by no means least, I would like to thank everyone in the InfoSec Research Group at UCL. It was great living the *#phdlife* with all of you.

Grazie!

Luca

# Contents

**1 Introduction** **25**

1.1 Research Questions . . . . . 28

1.2 Thesis Contributions . . . . . 29

1.3 Thesis Structure . . . . . 30

1.4 Publications . . . . . 31

1.5 Further Contributions . . . . . 31

**2 Background** **33**

2.1 Cryptography . . . . . 33

2.1.1 Tools . . . . . 33

2.1.2 Assumptions . . . . . 34

2.1.3 Differential Privacy (DP) . . . . . 35

2.2 Succinct Data Structures . . . . . 40

2.3 Machine Learning . . . . . 42

2.3.1 Recommender systems . . . . . 42

2.3.2 Time-series data prediction . . . . . 43

2.3.3 Kernel k-means with random features . . . . . 43

2.3.4 Neural networks . . . . . 45

2.3.5 Collaborative learning . . . . . 49

**3 Related Work** **53**

3.1 Private Statistics . . . . . 53

3.1.1 Privacy-preserving aggregation . . . . . 53

3.1.2 Privacy and succinct data representation . . . . . 56
3.2 Privacy in Machine Learning . . . . . 57
3.2.1 Learning with privacy . . . . . 57
3.2.2 Private data release . . . . . 59
3.2.3 Membership inference attacks . . . . . 61
3.2.4 Other attacks on machine learning models . . . . . 63

**4 Efficient Privacy-Preserving Computation of Statistics 67**
4.1 Private Recommender Systems For Streaming Services . . . . . 68
4.1.1 Overview . . . . . 68
4.1.2 Protocol . . . . . 69
4.1.3 Prototype implementation . . . . . 73
4.1.4 Performance evaluation . . . . . 74
4.2 Private Aggregate Location Prediction . . . . . 77
4.3 Tor Hidden Services Statistics . . . . . 82
4.3.1 Private median estimation using Count Sketch . . . . . 82
4.3.2 Implementation and evaluation . . . . . 85

**5 Privacy-Preserving Data Release with Generative Neural Networks 91**
5.1 Differentially Private Generative Model (DPGM) . . . . . 92
5.1.1 Private kernel k-means . . . . . 94
5.1.2 Private Stochastic Gradient Descent . . . . . 99
5.1.3 Adaptive selection of the norm bound . . . . . 99
5.1.4 Synthetic data generation . . . . . 101
5.2 Privacy Analysis . . . . . 102
5.3 Experimental Evaluation . . . . . 106
5.3.1 Experimental setup . . . . . 106
5.3.2 Results with image dataset . . . . . 107
5.3.3 Results with CDR and transit dataset . . . . . 112
5.3.4 Multi-layer Variational Autoencoder . . . . . 113

**6 Evaluating Privacy Leakage of Generative Models 115**

6.1 Attacks Outline . . . 116

6.1.1 Threat model . . . 116

6.1.2 White-box attack . . . 117

6.1.3 Black-box attack with no auxiliary knowledge . . . 119

6.1.4 Black-box attack with limited auxiliary knowledge . . . 120

6.2 Evaluation . . . 122

6.2.1 Experimental setup . . . 122

6.2.2 Naïve approaches . . . 124

6.2.3 White-box attack . . . 126

6.2.4 Black-box attack with no auxiliary knowledge . . . 128

6.2.5 Black-box attack with limited auxiliary knowledge . . . 129

6.2.6 Analysis . . . 133

6.2.7 Evaluation on Diabetic Retinopathy dataset . . . 134

6.3 Discussion . . . 136

6.3.1 Cost of the attacks . . . 137

6.3.2 Sensitivity to training set size and prediction ordering . . . 138

6.3.3 Defenses . . . 140

**7 Evaluating Privacy Leakage of Collaborative Learning 143**

7.1 Inference Attacks . . . 144

7.1.1 Threat model . . . 144

7.1.2 Overview of the attacks . . . 145

7.1.3 Membership inference . . . 146

7.1.4 Passive property inference . . . 147

7.1.5 Active property inference . . . 149

7.2 Experiments . . . 150

7.2.1 Datasets and model architectures . . . 150

7.2.2 Two-party membership inference . . . 153

7.2.3 Two-party single-batch property inference . . . 154

7.2.4 Inferring when a property occurs . . . 157

7.2.5 Active property inference . . . . . 159
7.2.6 Multi-party with synchronized SGD . . . . . 160
7.2.7 Multi-party with model averaging . . . . . 161
7.3 Defenses . . . . . 164
7.3.1 Selective gradient sharing . . . . . 164
7.3.2 Dimensionality reduction . . . . . 165
7.3.3 Dropout . . . . . 166
7.3.4 Participant-level differential privacy . . . . . 166
7.3.5 Sensitivity to number of training epochs . . . . . 167

**8 Conclusion** **169**

**Bibliography** **173**

# List of Figures

2.1 Update procedure for the Count-Min Sketch. . . . . 40

2.2 Generative Adversarial Network (GAN). . . . . 48

2.3 Collaborative Learning. . . . . 49

3.1 Samples from a GAN attack on a gender classification model where the class is "female". . . . . 64

4.1 Cryptographic layer of our private recommender system for online streaming services. At setup **(1)**, `users` compute their secret share and send their public key to the `tally`, who broadcasts them to the other `users`. During the encryption phase **(2)**, each `user` computes the blinding factors, encrypts their Count-Min Sketch and sends it to the `tally`. In case not all `users` have sent the data, the `tally` broadcasts $\mathcal{U}^{on}$, the subset of `users` that did **(3)**. These compute new blinding factors and send them to the `tally` **(4)**. Aggregate sketches are then recovered by the `tally` **(5)**. . . . . 70

4.2 Execution time for increasing number of `users` (with 700 programs). . . . . 74

4.3 Execution time for increasing number of programs (with 1,000 `users`). . . . . 75

4.4 Execution time for increasing number of programs (with 1,000 `users`) *without* Count-Min Sketch. . . . . 75

4.5 Visualizing the accuracy of the Count-Min Sketch for the most 50 frequent items (with 700 programs and sketch size 4,896). . . 78
4.6 Number of taxi locations over time. . . . . 79
4.7 Average error introduced by the Count-Min Sketch on the aggregate statistics for the top-100 locations. . . . . 80
4.8 Mean absolute error in the prediction for different values of prediction algorithm's parameter $\alpha$. . . . . 81
4.9 Mean absolute error introduced by the Count-Min Sketch on the prediction accuracy. . . . . 81
4.10 Count Sketch size versus estimation quality. . . . . 86
4.11 Quality versus differential privacy protection. . . . . 87

5.1 Overview of our differentially private generative model (DPGM) algorithm. . . . . 92
5.2 $\varepsilon$ value as a function of the number of SGD training epochs for MNIST ($\delta = 10^{-5}, T_{\mathcal{K}} = 20$) . . . . . 108
5.3 Clustering accuracy as a function of $\varepsilon$ on MNIST ($\delta = 10^{-5}, T_{\mathcal{K}} = 20$). . . . . 109
5.4 Real MNIST samples and samples generated from DPGM with RBM and VAE after 20 epochs ($\varepsilon = 1.74, T_{\mathcal{K}} = 20$). In **(c)** and **(d)**, each row contains 8 samples generated from a cluster. . . . 110
5.5 Average relative error vs. $\varepsilon$ for the CDR dataset ($q = 2.2 \cdot 10^{-5}, \delta = 4.4 \cdot 10^{-6}$) . . . . . 111
5.6 Average relative error vs. $\varepsilon$ for the transit dataset ($q = 10^{-4}, \delta = 10^{-6}$) . . . . . 112
5.7 Samples generated from a double layer VAE after 20 epochs. Each row contains 8 samples generated from a cluster. . . . . 114
5.8 Average relative error with $\varepsilon = 1.0$ for the CDR and transit datasets. . . . . 114

6.1 High-level Outline of the White-Box Attack. . . . . 118

6.2 White-Box Prediction Method: The attacker inputs data-points to the Discriminator $D$ (1), extracts the output probabilities (2), and sorts them (3). . . . . 118

6.3 High-level overview of the (a) black-box attack with no auxiliary knowledge, and (b) *Discriminative* and (c) *Generative* black-box attack with limited auxiliary attacker knowledge. . . . . 119

6.4 Real samples. . . . . 122

6.5 Euclidean attack results for DCGAN target model trained on a random 10% subset of CIFAR-10 and LFW. . . . . 125

6.6 Black-box attack results with 10% auxiliary attacker training set knowledge used to train a DCGAN *shadow model* for DCGAN target model trained on a random 10% subset of LFW. . . . . 125

6.7 Accuracy of white-box attack with different datasets and training sets. . . . . 126

6.8 Accuracy of black-box attack on different datasets and training sets. . . . . 128

6.9 Membership inference accuracy using a discriminative model, when the attacker has knowledge of (i) 20% of the test set, or (ii) 30% of both training and test sets. In (i), randomly guessing the training set corresponds to 14% accuracy, in (ii), to 12% accuracy. . . . . 130

6.10 Black-box attack results with 20% attacker training set knowledge for DCGAN/DCGAN+VAE target models, trained on a random 10% subset of LFW, for different delays at which auxiliary knowledge is introduced into the attacker model training. . 131

6.11 Black-box results when the attacker has (a) knowledge of 20% of the training set or (b) 30% of the training set and test set. The training set is a random 10% subset of the LFW or CIFAR-10 dataset, and the target model is fixed as DCGAN. . . . . 132

6.12 Accuracy curves and samples at different stages of training on top ten classes from the LFW dataset, showing a clear correlation between higher accuracy and better sample quality. . . . . 133

6.13 Various samples from the real dataset, target model, and black-box attack using the DCGAN target model on LFW, top ten classes. . . . . 133

6.14 Real and generated diabetic retinopathy dataset samples. . . . . 135

6.15 Accuracy curves of attacks against a DCGAN target model on the Diabetic Retinopathy dataset. . . . . 136

6.16 Improvements over random guessing, in a black-box attack, as we vary the size of the training set, and consider smaller subsets for training set predictions. . . . . 138

6.17 Improvement over random guessing for Weight Normalization and Dropout defenses against white-box attacks on models trained over different number of classes with LFW. . . . . 141

6.18 Accuracy curve and samples for different privacy budgets on top ten classes from the LFW dataset, showing a trade-off between samples quality and privacy guarantees. . . . . 142

7.1 Overview of inference attacks against collaborative learning. . . 145

7.2 Active property inference attack. . . . . 149

7.3 t-SNE projection of the features from different layers of the joint model on LFW gender classification; 0 is female, 1 is male. The property (i.e., the blue points denoted by p-0 and p-1) is “race: black”, while the red points without the property are denoted by np-0 and np-1 . . . . . 156

7.4 AUC vs. the fraction of the batch that has the property on FaceScrub and Yelp-author. . . . . 158

7.5 Detecting occurrence of a single-batch property. . . . . 158

7.6 Active property inference attack on FaceScrub. . . . . 160

7.7 Multi-party with synchronized SGD: attack AUC score vs. the number of participants. . . . . 161

7.8 Multi-party collaborative training with model averaging: box plots show the distribution of the adversary's scores in each trial. In the 8 trials on the left, one of the participants' data has the property; in the 8 trials on the right, none of the honest participants have the data with the property. . . . . 162

7.9 Detecting when a participant whose local data has the property of interest joins the training. $K = 2$ for rounds 0 to 250, $K = 3$ for rounds 250 to 500. . . . . 164

7.10 Uniqueness of user profiles with respect to the number of top locations. . . . . 164

7.11 Attack performance with respect to the number of collaborative learning epochs. . . . . 168

# List of Tables


4.1 Bytes exchanged by `user` and `tally` for different #`users` and size of the Count-Min Sketch, considering 700 programs. . . . . 77

4.2 Median estimation with 22 ciphertexts ($d = 2,\ w = 11,\ \epsilon, \delta = 0.25$) and 165 ciphertexts ($d = 3,\ w = 55,\ \epsilon, \delta = 0.05$) on the London Atlas Dataset. . . . . 89

5.1 Notation and symbols used in this chapter. . . . . 93

5.2 The datasets used in our experiments: MNIST (images), CDR (call detail records), and TRANSIT (transport records). . . . . 106

6.1 Accuracy of the best attacks on random 10% training set for LFW and CIFAR-10, and for the diabetic retinopathy (DR) experiments. . . . . 136

7.1 Datasets and tasks used in our experiments. . . . . 151

7.2 Precision of membership inference (recall is 1). . . . . 153

7.3 AUC score of single-batch property inference on LFW. We also report the Pearson correlation between the main task label and the property label. . . . . 155

7.4 Words with the largest positive coefficients in the property classifier for Yelp-health. . . . . 157

7.5 Inference attacks against the CSI Corpus for different fractions of gradients shared during training. . . . . 165

7.6 Membership inference against the CSI Corpus and FourSquare for different vocabulary sizes. . . . . 166

7.7 Inference of the top region (Antwerpen) on the CSI Corpus for different values of dropout probability. . . . . 167

# List of Algorithms


1 Parameter server with synchronized SGD . . . . . 50

2 Federated learning with model averaging . . . . . 51

3 DPGM: Differentially Private Generative Model . . . . . 94

4 DPkmeans: Private kernel $k$-means with Random Fourier Features . . . . . 96

5 Private Stochastic Gradient Descent . . . . . 100

6 DPNorm: Private Approximation of Average Norm . . . . . 101

7 Batch Property Classifier . . . . . 147

# Chapter 1

# Introduction

With the widespread deployment of complex Internet-based services, large amounts of data, including sensitive information, are being constantly published, collected, and processed [45, 117]. This abundance of contextual information makes it increasingly possible to extract value and knowledge from data. Examples include tracking GPS locations reported by mobile devices to generate live traffic maps (Google Traffic) and suggest more efficient routes [40]; analyzing social media content for disaster management [206], consumer confidence [161], and urban neighborhoods [181]; tracking disease incidence through geographic analysis of web search queries on the use of specific key words over time [121]; generating artificially created multimedia content from real data, e.g., images [112] and videos [182]; enabling endpoint devices to jointly learn a common predictive model while keeping all the raw data in the device [143].

This dissertation focuses on the privacy challenges presented by two novel and demand-driven technological trends in data processing: (1) *data collection*, which is the process of gathering information from different input sources, and (2) *machine learning* (ML), which gives artificial intelligence systems the capability to acquire their own knowledge by extracting patterns from raw data [86].

First, the large-scale collection of user data raises serious privacy, confidentiality, and liability concerns, thus motivating the need for efficient and scalable techniques allowing providers to *privately* gather statistics. Rather than re-

leasing only specific aggregate statistics, such as certain counting queries or histograms, entities are often willing or compelled to publish their datasets, e.g., aiming to monetize it or allow third parties with the appropriate expertise to analyze it. For instance, Call Detail Records (CDRs) collected by telecommunication companies are not only useful to capture interactions between customers, but also to understand their behavior, e.g., for infectious disease spreading or migration patterns.[1] As a result, telecommunications companies are often interested in releasing them in the form of "anonymized" datasets, which replace the original records in any data analytics without requiring any further interaction with the publisher.

However, as individuals usually have a unique combination of attribute values in these high-dimensional datasets, their exploitation and sharing are hindered by potential privacy breaches as well as implied monetary penalties. For instance, AOL released a detailed search logs dataset for research purposes, and Netflix released users' ratings to allow an open competition for the best ML algorithm designed to predict user ratings for unseen movies. Personal identifiers such as names were removed from these datasets as a guarantee for the users in the datasets to remain anonymous. Nonetheless, Narayanan and Shmatikov [157] were later able to identify individual users by cross-referencing their data records, and both companies were sued for privacy breach.[2,3]

Second, over the past few years, ML has played a increasing role in data processing systems due to its capability of efficiently discovering valuable knowledge and hidden information. Companies like Google, Microsoft, and Amazon provide customers with access to APIs that allow them to easily embed ML tasks into their applications. For instance, organizations provide *Machine Learning as a Service* (MLaaS) engines to outsource complex tasks, such as training classifiers, performing predictions, clustering, etc. They can also let other users query models trained on their data, possibly at a cost.

[1] See, e.g., `http://www.flowminder.org`

[2] `https://www.wired.com/2009/12/netflix-privacy-lawsuit/`

[3] `https://www.cnet.com/news/aol-sued-over-web-search-data-release/`

Among these techniques, *generative models* are an important emerging area in ML, as recent developments are paving the way for artificial generation of perfectly plausible images and videos. They are used in a number of applications, e.g., compression [198], denoising [23], inpainting [221], super-resolution [125], semi-supervised learning [183], clustering [197], and deep neural networks pre-training [79] in cases where labeled data is expensive.

However, if malicious users can infer sensitive properties of data used to train these models, this leads to dangerous information leakage. More specifically, the ability of an adversary to ascertain the presence of a specific data point in a training dataset constitutes an immediate privacy threat if the dataset is sensitive per se. For instance, if a model was trained on the records of patients with a certain disease, learning that an individual's record was part of the training data directly affects their privacy. On the one hand, users do not have much control over the kind of models and training parameters used by MLaaS platforms, and this might lead to overfitting (i.e., the model does not generalize well outside the data on which it was trained), thus making it easier for an attacker to perform inference attacks. On the other hand, this type of inference can also help enforce individual rights such as the "right to be forgotten", demonstrate inappropriate uses of data (e.g., the use of healthcare records to train ML models for unauthorized purposes [20]), and/or detect violations of data protection regulations such as the GDPR [81].

More recently, collaborative ML has emerged as an alternative to conventional MLaaS methodologies where all training data is pooled and the model trained on this joint pool. It allows two or more participants, each with his own training dataset, to work together to construct a joint model. More specifically, each participant trains a local model on his own data and periodically exchanges model parameters, updates to these parameters, or partially constructed models with the other participants. Many architectures, systems, and protocols have been proposed for distributed, collaborative, and federated learning [63, 51, 215, 154, 129, 228]: with and without a central server, with

different ways of aggregating and averaging models, etc. Their typical goal is to improve the training speed and reduce overheads, but protecting the privacy of participants' training data is an important motivation for several recent collaborative learning systems [143, 190]. Because the training data never leaves the participants' machines, collaborative learning may be considered as a good match for the scenarios where this data is sensitive (e.g., health-care records, private communications, personally identifiable information, etc.), and the participants want to construct a joint model without disclosing their datasets. Collaborative training, however, does disclose information via model updates that are indirectly based on the training data.

Together, these scenarios prompt a number of crucial challenges, including how to guarantee that the collected or published data are not misused; how to ensure that data processing does not lead to disclosure of sensitive information; and how to define privacy protection frameworks that allow usable and scalable services.

## 1.1 Research Questions

The broad goal of this dissertation is to tackle the following research problem:

| |
|---|
| Can we build and evaluate systems geared for privacy-friendly data processing, while enabling computational scenarios and applications where potentially sensitive data is needed to extract useful knowledge with strong privacy guarantees? |

Such goal entails addressing several open research questions, including:

1. Training ML models based on aggregate statistics gathered from many data sources without disclosing fine-grained information about single sources and in an efficient manner.

2. Releasing synthetic datasets that resemble real datasets without incurring privacy breaches.

3. Evaluating if an adversary can infer information about the data used to train ML models.

## 1.2 Thesis Contributions

Overall, this dissertation investigates the design and evaluation of privacy-aware data processing mechanisms. The contributions of this dissertation include:

1. We combine privacy-preserving aggregation with data structures supporting succinct data representation, namely, *Count-Min Sketch* [56] and *Count Sketch* [44]. Private aggregation is performed over the sketches, rather than the raw inputs. While an upper-bounded error in the aggregate is introduced, this allows us to reduce communication and computational complexity (for the cryptographic operations) from *linear* to *logarithmic* in the size of the inputs. We then use the resulting private statistics tools to instantiate protocols and build systems addressing real-world applications, where the error does not affect the overall quality of the computation. Specifically, we present and implement three protocols, (i) a privacy-preserving recommender system for on-line broadcasters, (ii) a private location prediction service, and (iii) a scheme for computing median statistics of Tor [65] hidden services in a private way.

2. We propose a novel approach, relying on generative neural networks, to model the data generating distribution of various kinds of data. It provides differential privacy [75] to each individual in the training data, thus, it can be used to effectively "anonymize" and share large high-dimensional datasets with any potentially adversarial third party. To this end, we present a Differentially Private Generative Model (DPGM), where data is first clustered, using the differentially private kernel k -means, and then each cluster is given to separate generative neural networks which are trained only on their own cluster using differentially private gradient descent.

3. We study how generating synthetic samples through generative models may lead to information leakage, hence, to violating privacy of individuals contributing their (sensitive) data to train these models. More specifically, given access to a generative model and an individual data record, we assess whether an attacker can tell if a specific record was used to train the model; this is also known as *membership inference*. Aiming to perform membership inference on generative ML models, we use Generative Adversarial Network (GAN) [87] models as a method to learn information about the target generative model, and thus create a local copy of the target model from which we can launch the attack.

4. We demonstrate that the training data used by participants in collaborative learning is vulnerable to a number of inference attacks. First, we show that an adversarial participant can infer the presence of exact data points of another participant's training data (i.e., membership inference). We then propose passive and active *property inference* attacks. These allow an adversarial participant in collaborative learning to infer properties of another participant's training data that are not true of the class as a whole, or even independent of the features that characterize the classes of the joint model.

## 1.3 Thesis Structure

The remainder of this dissertation is organized as follows. Chapter 2 provides background about notions and main tools that are used throughout the dissertation. Then, Chapter 3 discusses relevant related work in the context of privacy-preserving data processing. Chapters 4 to 7 contain the technical contribution of this dissertation. In particular, Chapter 4 covers work done on privacy-preserving collection of statistics. Then, Chapter 5 presents the work done on the problem of automating the process of private data release. In Chapter 6, we tackle the problem of evaluating privacy leakage in generative ML models. Chapter 7 covers our work on the topic of inference attacks against

collaborative ML.

Finally, Chapter 8 concludes the dissertation with a discussion on our contributions, and offers some potential future directions.

## 1.4 Publications

The material in this dissertation has been submitted or published in conferences and journals, co-authored with several researchers. Specifically, work in Chapter 4 has been done in collaboration with George Danezis and Emiliano De Cristofaro, and published in the proceedings of ISoc NDSS 2016 [147]. Chapter 5 presents the results of joint work with Gergely Acs, Claude Castelluccia, and Emiliano De Cristofaro, and published in the proceedings of IEEE ICDM 2017 and in the IEEE TKDE journal [3]. Chapter 6 is the outcome of the collaborative work with Jamie Hayes, George Danezis, and Emiliano De Cristofaro, and published in the journal Proceedings on Privacy Enhancing Technologies (PoPETs) 2019 [94]. Finally, Conghzeng Song, Vitaly Shmatikov, and Emiliano De Cristofaro have collaborated on the results presented in Chapter 7, and published in the proceedings of IEEE Security & Privacy 2019 [148].

## 1.5 Further Contributions

Besides the research included in this dissertation, we made further contributions with other researchers in the area of private network data processing. The associated research works have been published in the proceedings of the ACM CODASPY Workshop on SDN-NFV Security 2016 [149] and ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization [13], in collaboration with Hassan Jameel Asghar, Mohamed Ali Kaafar, Cyril Soldani, Emiliano De Cristofaro, and Laurent Mathy.

# Chapter 2

# Background

This chapter reviews concepts and tools in cryptography, succinct data structures, and machine learning that will be used throughout this dissertation.

## 2.1 Cryptography

We start by outlining some cryptographic primitives used in the rest of this dissertation.

### 2.1.1 Tools

**Negligible function.** A function $f(\tau)$ is negligible in the security parameter $\tau$ if, for every polynomial p, it holds that $f(\tau) < \frac{1}{|p(t)|}$, for large enough t.

**Pairwise Independent Hash Functions.** Let $\mathcal{H}$ be a family of *random-looking* hash functions mapping values from a domain [D] to a range [R]. $\mathcal{H}$ is *pairwise independent* if and only if $\forall x \neq y \in [D]$ and $\forall a_1, a_2 \in [R]$:

$$\Pr_{h \in \mathcal{H}}[h(x) = a_1 \wedge h(y) = a_2] = \frac{1}{R^2}.$$

**Fully Homomorphic Encryption (FHE)** A fully homomorphic encryption scheme (FHE) is a semantically secure cryptosystem that permits algebraic manipulations on plaintexts given their respective ciphertexts. More formally, a FHE scheme involves the following algorithms:

- *Key generation:* Given the security parameter $k$, generates public and private key pair $(pk, sk)$.

- *Encryption:* Given plaintext $m \in \{0,1\}^*$, outputs ciphertext $c = E(m)$ encrypted under public key $pk$.
- *Decryption:* Given a ciphertext $c$, outputs the plaintext $m = D(c)$ using the secret key $sk$.
- *Homomorphic Addition (Add):* Given two ciphertexts $c_1 = E(m_1)$, $c_2 = E(m_2)$, and the public key $pk$, produces a ciphertext $c = \text{Add}(c_1, c_2) = c_1 + c_2$ such that $D(c) = m_1 + m_2$.
- *Homomorphic Multiplication (Mult):* Given two ciphertexts $c_1 = E(m_1)$, $c_2 = E(m_2)$, and the public key $pk$, produces a ciphertext $c$ as $c = \text{Mult}(c_1, c_2) = c_1 \cdot c_2$ such that $D(c) = m_1 \cdot m_2$.

A partial homomorphic encryption scheme only supports either addition or multiplication.

Homomorphic encryption schemes allow arbitrarily computations to be performed on encrypted data without decrypting it. For instance, homomorphic encryption can be used to process encrypted DNA sequences in the cloud, or full text searching on encrypted data.

### 2.1.2 Assumptions

We now present some cryptographic assumptions used in the rest of this dissertation.

**Computational Diffie Hellman Assumption (CDH).** Let $\mathbb{G}$ be a cyclic group of order $q$ ($|q| = \tau$, for security parameter $\tau$), with generator $g$. We say that the Computational Diffie Hellman (CDH) problem is hard if, for any probabilistic polynomial-time algorithm $\hat{\mathcal{A}}$ and random $x, y$ drawn from $\mathbb{Z}_q$:

$$\Pr\left[\hat{\mathcal{A}}(\mathbb{G}, q, g, g^x, g^y) = g^{xy}\right]$$

is negligible in the security parameter $\tau$.

**Decisional Diffie Hellman Assumption (DDH).** Let $\mathbb{G}$ be a cyclic group of order $q$ ($|q| = \tau$), with generator $g$. We say that the Decisional Diffie Hellman

(DDH) problem is hard if, for any probabilistic polynomial-time algorithm $\hat{\mathcal{A}}'$ and random $x, y, z$ drawn from $\mathbb{Z}_q$:

$$\left| \Pr\left[\hat{\mathcal{A}}'(\mathbb{G}, q, g, g^x, g^y, g^z) = 1\right] - \Pr\left[\hat{\mathcal{A}}'(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1\right] \right|$$

is negligible in the security parameter $\tau$.

These assumptions are based on problems involving discrete logarithms in cyclic groups. They are commonly used as the basis to prove the security of many cryptographic protocols, e.g., the ElGamal cryptosystem [78]. The DDH and CDH assumptions are related to each other. If it were possible to efficiently compute $g^{xy}$ from $(g^x, g^y)$, then one could easily distinguish the two probability distributions in DDH. It is believed that DDH is a stronger assumption than CDH. DDH and CDH are variants of the more general Diffie Hellman problem (DHP) in which, given $g$, $g^x$ and $g^y$, the problem is to find the value of $g^{xy}$. The main motivation for this problem is that many security systems rely on one-way functions, i.e., operations that are fast to compute, but hard to reverse. In cryptography, one-way functions enable encrypting a message, while making it difficult to decrypt the same message without knowing some private information. If it were possible to efficiently solve DHP, then security systems that rely on DHP would be easily broken.

### 2.1.3 Differential Privacy (DP)

Differential privacy can be motivated by the impossibility result, for a statistical database, to achieve the privacy goal of preventing disclosure about any individual against adversaries with arbitrary auxiliary while still providing any useful information. We can further consider the following example taken from [69]:

Given a statistical database that provides the average height for population subgroups, and the auxiliary information "Terry Gross is two inches shorter than the average Lithuanian woman". An adversary with this auxiliary information and access to the database is able to recover Terry Gross' height, whereas an adversary with only auxiliary information learns much less about

Terry Gross' height. However, Terry Gross is not required to be part of the statistical database in order for this privacy breach to happen. Therefore, differential privacy notion aims to *minimize the increased risk* that an individual incurs by joining – or leaving – the database [69].

More concretely, differential privacy allows a party to privately release a dataset: using perturbation mechanisms, a function of an input dataset is modified, so that any information which can discriminate a record from the rest of the dataset is bounded [75]. Hence, any information that can be learned from the database with a record can also be learned from the one without this record. Consequently, for a record owner, it means that any privacy breach will not be due to participating in the database.

More formally, $\varepsilon$-differential privacy is defined as follows:

**Definition 1** ($\varepsilon$-Differential Privacy [75])**.** *A privacy mechanism $\mathcal{A}$ guarantees $\varepsilon$-differential privacy if for any database $X \in \mathcal{X}$ and $X' \in \mathcal{X}$ differing on at most one record, and for any possible output $O \in Range(\mathcal{A})$,*

$$e^{-\varepsilon} \times Pr[\mathcal{A}(X') = O] \leq Pr[\mathcal{A}(X) = O] \leq e^{\varepsilon} \times Pr[\mathcal{A}(X') = O]$$

*where the probability is taken over the randomness of $\mathcal{A}$.*

Intuitively, this guarantees that an adversary, provided with the output of $\mathcal{A}$, can draw almost the same conclusions about any individual no matter if this individual is included in the input of $\mathcal{A}$ or not [75]. The following definition of *privacy loss* can then be formally derived.

**Definition 2** (Privacy loss)**.** *Let $\mathcal{A}$ be a privacy mechanism which assigns a value $O \in Range(\mathcal{A})$ to a dataset $X \in \mathcal{X}$. The privacy loss of $\mathcal{A}$ with datasets $X \in \mathcal{X}$ and $X' \in \mathcal{X}$ at output $O \in Range(\mathcal{A})$ is a random variable $\mathcal{P}(\mathcal{A}, X, X', O) = \log \frac{\Pr[\mathcal{A}(X)=O]}{\Pr[\mathcal{A}(X')=O]}$ where the probability is taken on the randomness of $\mathcal{A}$.*

A relaxation of DP is probabilistic-DP, or $(\epsilon, \delta)$-DP, where privacy breaches may occur with very small probability.

**Definition 3** ($(\epsilon, \delta)$-Differential Privacy [75]). *A privacy mechanism $\mathcal{A}$ guarantees $(\varepsilon, \delta)$-differential privacy if for any database $X \in \mathcal{X}$ and $X' \in \mathcal{X}$, differing on at most one record, and for any possible output $S \subseteq Range(\mathcal{A})$,*

$$Pr[\mathcal{A}(X) \in S] \leq e^{\varepsilon} \times Pr[\mathcal{A}(X') \in S] + \delta$$

*or, equivalently,*

$$\Pr_{O \sim \mathcal{A}(X)}[\mathcal{P}(\mathcal{A}, X, X', O) > \varepsilon] \leq \delta.$$

This definition guarantees that every output of algorithm $\mathcal{A}$ is almost equally likely (up to $\varepsilon$) on datasets differing in a single record except with probability at most $\delta$, preferably smaller than $1/|X|$. Probabilistic-DP provides higher utilities in practice than $\varepsilon$-DP (Definition 1) at the cost of weaker privacy guarantees.

A fundamental concept in DP is the *global sensitivity* of a function [75].

**Definition 4** (Global $L_p$-sensitivity). *For any function $f : \mathcal{X} \to \mathbb{R}^d$, the $L_p$-sensitivity of $f$ is $\Delta_p f = \max_{X,X'} ||f(X) - f(X')||_p$, for all $X, X'$ differing in at most one record, where $||\cdot||_p$ denotes the $L_p$-norm.*

There are a few ways to achieve DP, including the Laplace mechanism and the Gaussian mechanism. The Laplace mechanism (LPM) consists of adding noise sampled from the Laplace distribution to the true output of a function.

**Definition 5** (Laplace mechanism (LPM) [75]). *For any function $f : \mathcal{X} \to \mathbb{R}^d$, LPM is defined as*

$$\mathcal{LP}(X) = f(X) + [\hat{\mathcal{Y}}_1(\hat{s}), \ldots, \hat{\mathcal{Y}}_d(\hat{s})]$$

*where $\hat{\mathcal{Y}}_i(\hat{s})$ are i.i.d Laplace random variables with scale parameter $\hat{s} = \Delta_1 f/\varepsilon$, and $\Delta_1 f$ is the $L_1$-sensitivity of $f$.*

It can be proved that the above definition of LPM achieves $\varepsilon$-DP [75]. Instead, the Gaussian Mechanism (GM) consists of adding gaussian noise to the true output of a function.

**Definition 6** (Gaussian Mechanism (GM) [75]). *For any function $f : \mathcal{X} \to \mathbb{R}^d$, GM is defined as*

$$\mathcal{G}(X) = f(X) + [\mathcal{N}_1(0, \Delta_2 f \cdot \sigma), \ldots, \mathcal{N}_d(0, \Delta_2 f \cdot \sigma)]$$

*where $\mathcal{N}_i(0, \Delta_2 f \cdot \sigma)$ are i.i.d. normal random variables with zero mean and variance $(\Delta_2 f \cdot \sigma)^2$, and $\Delta_2 f$ is the $L_2$-sensitivity of $f$.*

The above definition of GM is $(\varepsilon, \delta)$-DP if $\sigma \geq \Delta_2 f / \varepsilon$ for $c^2 > 2\ln(1.25)/\delta$ [75].

The output of any randomized algorithm remains differentially private if all inputs are already differentially private. This is often referred to as the *post-processing property* of DP. Further, DP maintains composition.

**Theorem 1** (Composition property of DP [145]). *Let $\mathcal{A}_i$ each provide $\varepsilon_i$-differential privacy. It holds:*

1. *A sequence of $\mathcal{A}_i(X)$ over the dataset $X$ provides $\sum_i \varepsilon_i$-differential privacy.*
2. *A sequence of $\mathcal{A}_i(X_i)$ over a set of disjoint[1] datasets $X_i$ provides $\max(\varepsilon_i)$-differential privacy.*

In the case of probabilistic-DP, if each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ is $(\varepsilon, \delta)$-DP, then their $k$-fold adaptive composition[2] is $(k\varepsilon, k\delta)$-DP. However, a tighter upper bound can be derived on the privacy loss of the composite using a generic Chernoff bound. In particular, it follows from Markov's inequality that

$$\Pr[\mathcal{P}(\mathcal{A}, X, X', O) \geq \varepsilon] \leq \mathbb{E}[\exp(\lambda \mathcal{P}(\mathcal{A}, X, X', O))] / \exp(\lambda\varepsilon)$$

for any output $O \in \mathit{Range}(\mathcal{A})$ and $\lambda > 0$. This implies that $\mathcal{A}$ is $(\varepsilon, \delta)$-DP with $\delta = \min_\lambda \exp(\beta_\mathcal{A}(\lambda) - \lambda\varepsilon)$, where $\beta_\mathcal{A}(\lambda) = \max_{X,X'} \log \mathbb{E}_{O \sim \mathcal{A}(X)}[\exp(\lambda \mathcal{P}(\mathcal{A}, X, X', O))]$ is the log of the moment generating function of the privacy loss.

[1] Two datasets are disjoint if they have no common records.

[2] The output of $\mathcal{A}_{i-1}$ is used as input to $\mathcal{A}_i$, i.e., their executions are not necessarily independent except their coin tosses.

This result is referred to as the *moments accountant* method, which we formally define as follows:

**Theorem 2** (Moments accountant (Abadi et al. [1]))**.** *Let* $\beta_{\mathcal{A}_i}(\lambda)$ *be*

$$\max_{X,X'} \log \mathbb{E}_{O\sim\mathcal{A}(X)}[\exp(\lambda \mathcal{P}(\mathcal{A}, X, X', O))]$$

*and* $\mathcal{A}_{1:k}$ *the k-fold adaptive composition of* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$*. It holds:*

1. $\beta_{\mathcal{A}_{1:k}}(\lambda) \leq \sum_{i=1}^{k} \beta_{\mathcal{A}_i}(\lambda)$
2. $\mathcal{A}_{1:k}$ *is* $(\varepsilon, \min_\lambda \exp(\sum_{i=1}^{k} \beta_{\mathcal{A}_i}(\lambda) - \lambda\varepsilon))$*-differentially private*

*where* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ *use independent coin tosses.*

Moreover, if an increase in the $\delta$ term is tolerated, the privacy parameter $\varepsilon$ degrades proportionally to $\sqrt{k}$, and the composite is $(\varepsilon \cdot O(\sqrt{k \log(1/\delta')}), k\delta + \delta')$-differentially private for all $k < 1/\varepsilon^2$ and $\delta' > 0$. This result is known as the advanced composition property of differential privacy [71].

Finally, we introduce the following useful Lemma.

**Lemma 1.** *Let* $\mathcal{G}$ *be the Gaussian Mechanism. It holds:* $\beta_{\mathcal{G}}(\lambda) = (\lambda^2 + \lambda)/4\sigma^2$

*Proof.* Let $f : \mathcal{X} \to \mathbb{R}$ be a scalar function, $f(X) = f(X') + \Delta_1 f$, where $\Delta_1 f = \Delta_2 f$, and $O = f(X) + x$, where $x \sim \mathcal{N}(0, \sigma)$. Let $\hat{\sigma} = \Delta_1 f \cdot \sigma$ Then, it holds:

$$\begin{aligned}
\mathcal{P}(\mathcal{A}, X, X', O) &= \ln\left(\frac{\Pr[\mathcal{G}(X) = O]}{\Pr[\mathcal{G}(X') = O]}\right) = \\
&= \ln\left(\frac{\Pr[f(X) + \mathcal{N}(0, \hat{\sigma}) = O]}{\Pr[f(X') + \mathcal{N}(0, \hat{\sigma}) = O]}\right) = \ln\left(\frac{\exp(-x^2/2\hat{\sigma}^2)}{\exp(-(x + \Delta_1 f)^2/2\hat{\sigma}^2)}\right) = \\
&= \ln\left(\frac{\exp(-x^2/2\hat{\sigma}^2)}{\exp(-(x + \Delta_1 f)^2/2\hat{\sigma}^2)}\right) = \left(\frac{\Delta_1 f}{\hat{\sigma}} \cdot \frac{x}{\hat{\sigma}}\right) + \frac{1}{2}\left(\frac{\Delta_1 f}{\hat{\sigma}}\right)^2
\end{aligned}$$

Since $x$ is drawn from $\mathcal{N}(0, \hat{\sigma})$, $\mathcal{P}(\mathcal{A}, X, X', O)$ follows a normal distribution with mean $(\Delta_1 f)^2/2\hat{\sigma}^2$ and standard deviation $\Delta_1 f/\hat{\sigma}$, whose moment generating function is $\exp\left((\lambda^2 + \lambda)(\Delta_1 f)^2/4\hat{\sigma}^2\right)$. The claim follows from the definition of $\beta$ and $\hat{\sigma}$. For the high-dimensional case when $f : \mathcal{X} \to \mathbb{R}^d$ $(d > 1)$, the proof is similar to that of Theorem A.1 in [75].

□

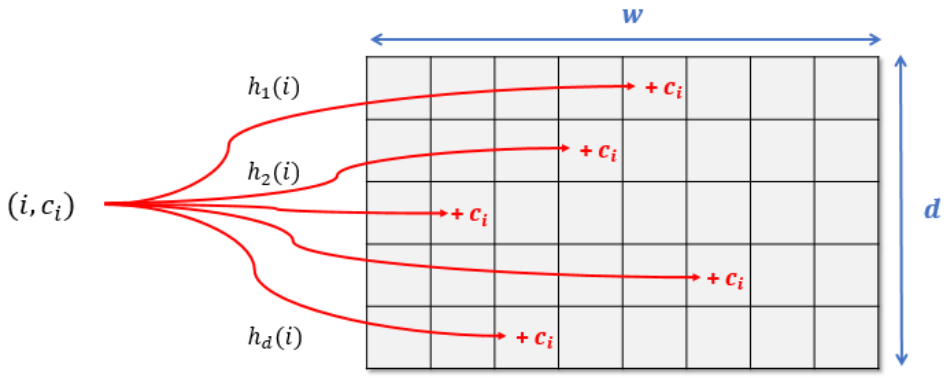



**Figure 2.1:** Update procedure for the Count-Min Sketch.

Given $\beta_{\mathcal{G}}(\lambda)$, the exact privacy cost $\varepsilon$ (or $\delta$) of the $k$-fold adaptive composition of $\mathcal{G}$ is computed based on Theorem 2.

## 2.2 Succinct Data Structures

This section introduces some succinct data structures, namely Count-Min Sketch and Count Sketch, for efficient query operations. These "sketching" data structures use minimal space to allow estimating the most frequent items in a large set of data. They are used in a number of applications, e.g., tracking Twitter's most popular tweets, or counting the most visited pages of a website.

**Count-Min Sketch [56]** is a data structure that can be used to provide a succinct sublinear-space representation of multi-sets. An interesting property is that they enable aggregation of the multi-sets represented by two or more sketches using a linear operation on the sketches themselves. Prior uses of Count-Min Sketch include summarizing large amounts of frequency data for sensing, networking, natural language processing, and database applications [62].

**Definition 7** (*Count-Min Sketch*)**.** *A Count-Min Sketch with parameters* $(\epsilon, \delta)$ *is a two-dimensional array (table)* $\tilde{X}$*, with width* $w$ *and depth* $d$*. Given parameters* $(\epsilon, \delta)$*, set* $d = \lceil \ln T/\delta \rceil$ *and* $w = \lceil e/\epsilon \rceil$*, where* $T$ *is the number of items to be counted. Each entry of the table is initialized to zero. Then,* $d$ *hash functions* $h_j : \{0,1\}^* \to \{0,1\}^w$*, are chosen uniformly at random from a*

*pairwise-independent family* $\mathcal{H}$.

*Update Procedure.* To update item $i$ by a quantity $c_i$, $c_i$ is added to one element in each row, where the element in row $j$ is determined by the hash function $h_j$. The update is denoted as $(i, c_i)$. More precisely, to update the count for item $i$ to $c_i \in \mathbb{N}$, for each row $j$ of $\tilde{X}$, set:

$$\tilde{X}[j, h_j(i)] \leftarrow \tilde{X}[j, h_j(i)] + c_i$$

This procedure is illustrated in Figure 2.1.

*Estimation Procedure.* To estimate the count $\hat{c_i}$ for item $i$, we take the minimum of the estimates of $c_i$ from every row of $\tilde{X}$:

$$\hat{c_i} \leftarrow \min_j \tilde{X}[j, h_j(i)]$$

*Error Upper Bound.* Given estimate $\hat{c_i}$, it holds:

1. $c_i \leq \hat{c_i}$
2. $\hat{c_i} \leq c_i + \epsilon \sum_{j=1}^{T} |c_j|$ with probability $1 - \delta$.

(where $c_i$ is the true counter).

**Count Sketch [44]** is a data structure which provides an estimate for an item's frequency in a stream. Count Sketch has the same update procedure as Count-Min Sketch, but differs in the estimation. Specifically, given the table $\tilde{X}$ built on the stream, the row estimate of $c_i$ (which is the counter of item $i$) for row $j$ is computed based on two buckets: $\tilde{X}[i, h_j(i)]$ and $\tilde{X}[i, h'_j(i)]$, where $h'_j(i)$ is defined as:

$$h'_j(i) := \begin{cases} h_j(i) - 1 & \text{if } h_j(i) \bmod 2 = 0 \\ h_j(i) + 1 & \text{if } h_j(i) \bmod 2 = 1 \end{cases}$$

The estimate of $c_i$ for row j is then

$$\left(\tilde{X}[j, h_j(i)] - \tilde{X}[j, h'_j(i)]\right)$$

To estimate the count $\hat{c_i}$ for item $i$, we take the sum of the estimates of $c_i$ from every row of $X$:

$$\hat{c_i} \leftarrow \underset{j}{sum} \left(\tilde{X}[j, h_j(i)] - \tilde{X}[j, h'_j(i)]\right)$$

Both Count-Min and Count Sketch are linear: the element-wise sum of the sketches representing two multi-sets yields the sketch of their union.

Note that, although we use $(\epsilon, \delta)$ to denote parameters for both Count(-Min) Sketch and differential privacy (see Section 2.1.3), it is clear from the context which one it relates to.

## 2.3 Machine Learning

In this section, we review machine learning concepts used throughout the dissertation.

### 2.3.1 Recommender systems

Recommender systems are used to predict the utility of a certain item for a particular user, based on their previous ratings as well as those of other "similar" users [180]. Consider a set of $N$ users and a list of $M$ items: for each user, a rating can be associated to each item, based, e.g., on the user's explicit opinion about the item (e.g., 1 to 5 stars) or by implicitly deriving it from purchase records or browser history.

Machine learning can be used to predict the expected rating of an unrated item for a given user. The *K-Nearest Neighbor (KNN)* classification algorithm finds the top-$K$ nearest neighbors for a given item, so that ratings associated with these are combined to predict unknown ratings. A variant of KNN is called *ItemKNN* [185]. The algorithm is trained using an item-to-item similarity matrix (correlation matrix), where each element expresses the similarity between a pair of items, and the Cosine Similarity is computed between vectors of items (e.g., user ratings for each item).

If ratings are binary values (e.g., viewed/not viewed), the Cosine Similarity between items $a$ and $b$ is:

$$\{Sim\}_{ab} = \frac{C_{ab}}{\sqrt{C_a \cdot C_b}} \tag{2.1}$$

where $C_{ab}$, $C_a$, and $C_b$ denote, respectively, the number of people who rated both $a$ and $b$, $a$, and $b$. Given the similarity matrix, we can identify the nearest

neighbors for each item as the items with the highest correlation values. The final model then consists of the identity of the nearest neighbors and their correlation values (or *weights*) which are used in the prediction process, i.e., the items that should be recommended.

Note that, with ItemKNN, given the item-to-item matrix, each user could independently compare their ratings with the nearest neighbors of each item in the model. Upon finding a match, the weight is added to the prediction score for that item. The items are then ranked by their prediction scores and the top $K$ are taken as recommendations. Predicting user ratings or interests in general has many applications especially in e-commerce systems (e.g. Amazon Web store).

### 2.3.2 Time-series data prediction

Time-series data prediction can be performed, among others, using Exponential Weighted Moving Average (EWMA) models [192]. EWMA can predict future values based on past values weighted with exponentially decreasing weights toward older values. Given a signal over time $r(t)$, we indicate with $\tilde{r}(t+1)$ the predicted value of $r(t+1)$ given the past observations, $r(t')$, at time $t' \leq t$. Predicted signal $\tilde{r}(t+1)$ is estimated as:

$$\tilde{r}(t+1) = \sum_{t'=1}^{t} \alpha(1-\alpha)^{t-t'} r(t')$$

where $\alpha \in (0,1)$ is the smoothing coefficient, and $t' = 1,\ldots,t$ indicates the training window, i.e., 1 corresponds to the oldest observation while $t$ is the most recent one.

### 2.3.3 Kernel k-means with random features

Clustering is the task of grouping a set of objects in such a way that objects in the same group, or *cluster*, are more similar to each other than to those in other groups. $k$-means is one of the most popular clustering algorithms. Given a set of samples $X = \{x_1, x_2, \ldots, x_N\}$, $k$-means linearly separates $X$ into $k$ clusters $C_1, C_2, \ldots, C_k$ $(k \leq N)$ so that it aims to minimize the error

$\sum_{i=1}^{k}\sum_{x\in C_i}||x-c_i||_2^2$, where $c_i=\sum_{x\in C_i}x/|C_i|$ is the centroid of cluster $C_i$. Although this problem is NP-hard, there are efficient heuristic algorithms (such as Lloyd's algorithm) which iteratively refines clustering and converge quickly to a local optimum. However, $k$-means can provide very inaccurate clustering of linearly *non*-separable data, which are very common in practice.

To overcome this shortcoming, kernel $k$-means [187] first maps samples from input space to a higher dimensional feature space through a non-linear transformation $\Phi$, then applies standard $k$-means on $\{\Phi(x_1),\Phi(x_2),\ldots,\Phi(x_N)\}$. Hence, kernel $k$-means provides linear separators of clusters in feature space which correspond to non-linear separators in input space. Kernel $k$-means iteratively computes $||\Phi(x)-c_i'||_2^2$ for each sample $x$ to decide which cluster a sample belongs to, where $c_i'=\sum_{x\in C_i}\Phi(x)/|C_i|$. To do so, the inner product $\langle\Phi(x),\Phi(y)\rangle$ must be known for all $x,y\in X$. Since $\Phi(\cdot)$ is hard to explicitly compute due to its large, often infinite dimension, the kernel trick is applied; $\langle\Phi(x),\Phi(y)\rangle=\kappa(x,y)$, where $\kappa$ is an easily computable kernel function. Still, this approach requires evaluating $\kappa$ for all pairs of samples and store the results, which is not scalable for large datasets.

To make kernel $k$-means scalable, the kernel function can be approximated with low-dimensional explicit feature maps. In particular, the samples are first mapped to a low-dimensional Euclidean inner product space using an explicit random feature map $z:\mathbb{R}^m\to\mathbb{R}^d$ so that $\langle\Phi(x),\Phi(y)\rangle\approx\langle z(x),z(y)\rangle$. Then, standard $k$-means is applied on the low-dimensional mapped samples $\{z(x_1),z(x_2),\ldots,z(x_N)\}$ in $\mathbb{R}^d$ to approximate the result of the kernel $k$-means with implicit feature map $\Phi$ and kernel $\kappa$. Although the approximation error decreases as $d$ increases, quite accurate approximations can be obtained even for $d<m$. Explicit nonlinear feature maps have already been proposed for shift-invariant kernels (e.g., generalized RBF kernels) [207] as well as polynomial kernels [170] among others.

### 2.3.4 Neural networks

In recent years, a family of machine learning models known as deep neural networks (or *deep learning*) has become very popular for many machine learning tasks, especially related to computer vision and image recognition [186, 123]. Deep learning models are made of layers of non-linear mappings from input to intermediate hidden states and, finally, to output. Each connection between layers has a floating-point weight matrix as parameters. These weights are updated during training. The topology of the connections between layers is task-dependent and important for the ultimate accuracy of the model. To find the optimal set of parameters that fits the training data, the training algorithm optimizes the *objective (loss) function* $\mathcal{L}$, which penalizes the model when it outputs a wrong label on a data point.

There are many methods to optimize the objective function. *Stochastic gradient descent* (SGD) and its variants (e.g., Adam optimizer [116]) are commonly used to train artificial neural networks. SGD is an iterative method where at each step the optimizer receives a small batch of the training data and updates the model parameters $\theta$ according to the direction of the negative gradient of the objective function with respect to $\theta$ and scaled by the learning rate $\eta \in \mathbb{R}^+$. Training finishes when the model has converged to a local minimum, where the gradient is close to zero.

Machine learning models include discriminative and generative ones. Given a supervised learning task, and given the features of a data-point $x \in X$ and the corresponding label $y \in Y$, discriminative models attempt to predict $y$ on future $x$ by learning a discriminative function $f$ from $(x, y)$; the function takes in input $x$ and outputs the most likely label $y$. Discriminative models are not able to "explain" how the data-points might have been generated. By contrast, generative models describe how data is generated by learning the joint probability distribution of $p(X, Y)$, which gives a score to the configuration determined together by pairs $(x, y)$. Generative models based on deep neural networks, such as Generative Adversarial Networks (GAN) [87] and

Variational Auto-encoders (VAE) [115] are considered as the state-of-the-art for producing samples of realistic images [111].

We now briefly describe three generative neural networks models used in this dissertation in Chapters 5 and 6.

**Restricted Boltzmann Machines (RBM).** A Restricted Boltzmann Machine (RBM) is a bipartite undirected graphical model composed of $m$ visible and $n$ invisible (or latent) binary random variables denoted by, respectively, $v = (v_1, v_2, \ldots, v_m)$ and $h = (h_1, h_2, \ldots, h_n)$. Visible variables represent the attributes of $x$ and their values are composed of records from $X$. Hidden variables capture the dependencies between different visible variables. As the above model is a Markov random field with strictly positive joint probability distribution $\tilde{p}$ over the model variables, $\tilde{p}$ can be represented as a Boltzmann distribution defined as:

$$\tilde{p}(v,h) = \frac{1}{Z} e^{-E(v,h)} \tag{2.2}$$

where $Z = \sum_{v,h} e^{-E(v,h)}$ is the partition function, $E(v,h)$ the energy function, i.e., $E(v,h) = -\sum_{i=1}^{n}\sum_{j=1}^{m} \cdot W_{ij} \cdot h_i \cdot v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i$, with $W_{ij}$ being real valued weights describing the inter-dependency between $v_j$ and $h_i$, and $b_j, c_i$ real valued bias terms associated with the $j$th visible and $i$th hidden units, respectively. Using matrix notation, $E(v,h) = -v \cdot W \cdot h - b \cdot v - c \cdot h$, where $W = \{W_{i,j}\}$, $c = \{c_i\}$, and $b = \{b_j\}$. The goal is to approximate the true data generating distribution with the Boltzmann distribution $\tilde{p}$, given in Eq. (2.2). To this end, we train the RBM model on dataset $X$ to compute parameters $(W, b, c)$.

There are a few algorithms to train RBMs, that approximate or relate to gradient descent on the log-loss of the data. If $\theta = (W, b, c)$, then we want to minimize the loss function $\mathcal{L}(X;\theta) = -\prod_{x \in X} \tilde{p}(x|\theta)$ given dataset $X$, where $x \in \{0,1\}^m$ is a record from $X$ and $\tilde{p}$ is the Boltzmann distribution defined in Eq. (2.2). The model parameters are updated until the log-loss converges using Persistent Contrastive Divergence [199].

**Variational Autoencoder (VAE).** A variational autoencoder [115] consists of two neural networks (an *encoder* and a *decoder*), and a loss function. The encoder compresses data into a latent space $z$ while the decoder reconstructs the data given the hidden representation. Let $x$ be a random vector of $m$ observed variables, which are either discrete or continuous. Let $z$ be a random vector of $n$ latent continuous variables. The probability distribution between $x$ and $z$ assumes the form $p_\theta(x,z) = p_\theta(z)p_\theta(x \mid z)$, where $\theta$ indicates that $p$ is parametrized by $\theta$. Also, let $q_\phi(z \mid x)$ be a recognition model whose goal is to approximate the true and intractable posterior distribution $p_\theta(z \mid x)$. We can then define a lower-bound on the log-loss of $x$ as follows: $\mathcal{L}(x;\theta) = D_{KL}(q_\phi(z \mid x) \,||\, p_\theta(z)) - \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)]$. The first term pushes $q_\phi(z \mid x)$ to be similar to $p_\theta(z)$ ensuring that, while training, VAE learns a decoder that, at generation time, will be able to invert samples from the prior distribution such they look just like the training data. The second term can be seen as a form of reconstruction cost, and needs to be approximated by sampling from $q_\phi(z \mid x)$.

In VAEs, we propagate the gradient signal through the sampling process and through $q_\phi(z \mid x)$ using the *reparametrization trick*. This is done by making $z$ be a deterministic function of $\phi$ and some noise $\xi$, i.e., $z = f(\phi,\xi)$. For instance, sampling from a normal distribution can be done like $z = \mu + \sigma\xi$, where $\xi \sim \mathcal{N}(0, I)$. The reparametrization trick can be viewed as an efficient way of *adapting* $q_\phi(z \mid x)$ to help improve the reconstruction. We train the variational autoencoder using stochastic gradient descent to optimize the loss with respect to the parameters of the encoder and decoder $\theta$ and $\phi$.

**Generative Adversarial Network (GAN).** GANs [87] are neural networks trained in an adversarial manner to generate data mimicking some distribution. The main intuition is to have two competing neural network models. One model, the *generator*, takes noise as input and generates samples. The other model, the *discriminator*, receives samples from both the generator and the training data, and has to be able to distinguish between the two sources.

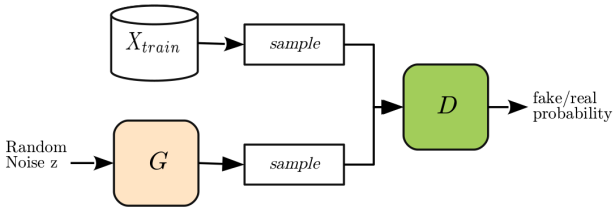



**Figure 2.2:** Generative Adversarial Network (GAN).

The two networks play a continuous game where the generator is learning to produce more and more realistic samples, and the discriminator is learning to get better and better at distinguishing generated data from real data, as depicted in Figure 2.2.

More formally, to learn the generator's output distribution over data-points $x$, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where $G$ is a generative deep neural network with parameters $\theta_g$. We also define a discriminator $D(x;\theta_d)$ that outputs $D(x) \in [0,1]$, representing the probability that $x$ was taken from the training set rather than from the generator $G$. $D$ is trained to maximize the probability of assigning the correct label to both real training examples and fake samples from $G$. We simultaneously train $G$ to minimize $\log(1-D(G(z)))$. The final optimization problem solved by the two networks $D$ and $G$ follows a two-player minimax game as:

$$\min_G \max_D \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1-D(G(z)))]$$

First, gradients of $D$ are computed to discriminate fake samples from training data, then $G$ is updated to generate samples that are more likely to be classified as data. After several steps of training, if $G$ and $D$ have enough capacity and a Nash equilibrium is achieved, they will reach a point at which both cannot improve [87].

Larsen et al. [122] combine VAEs and GANs into an unsupervised generative model that simultaneously learns to encode and generate new samples, which contain more details, sampled from the training data-points.

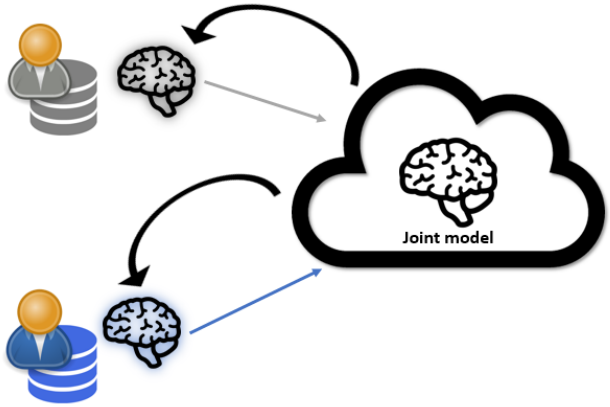



**Figure 2.3:** Collaborative Learning.

Recently, Lucic et al. [135] show that, despite a large number of proposed changes to the original GAN model [141, 25, 118, 12, 90], it is still difficult to assess if one performs better than another. They also show that the original GAN performs equally well against other state-of-the-art GANs, concluding that any improvements are due to computational budgets and hyper-parameter tuning, rather than scientific breakthroughs.

### 2.3.5 Collaborative learning

Training a deep neural network on a large dataset can take long time. A common approach to scale deep learning [51, 63] is to partition the training dataset, concurrently train separate models on each subset, and exchange parameters via a parameter server which stores the global set of all model parameters (see Figure 2.3). During training, each local model pulls the parameters from this server, calculates the updates based on its current batch of training data, then pushes these updates back to the server, which updates the global parameters.

Collaborative learning may involve participants who want to hide their training data from each other. We review two architectures for privacy-preserving collaborative learning based on, respectively, [190] and [143].

**Parameter server with synchronized gradient updates.** We consider collaborative learning with synchronized gradient updates, as proposed

**Algorithm 1:** Parameter server with synchronized SGD

**Server executes:**
  Initialize $\theta_0$
  **for** $t = 1$ *to* $T$ **do**
    **for** *each client* $k$ **do**
      $g_t^k \leftarrow$**ClientUpdate**$(\theta_{t-1})$
    $\theta_t \leftarrow \theta_{t-1} - \eta \sum_k g_t^k$ ▷ synchronized gradient updates

**ClientUpdate**$(\theta)$:
  Select a batch $B$ from client's data
  **return** local gradients $\nabla \mathcal{L}(B; \theta)$

in [190]: see Algorithm 1. In each iteration of training, the participants download the global model from the parameter server. Each participant then locally computes gradient updates based on one batch of his training data and sends the updates to the server. The server waits for the gradient updates from all participants and then applies the aggregated updates to the global model using stochastic gradient descent (SGD).

**Federated learning with model averaging.** We also consider the federated learning framework proposed in [143]: see Algorithm 2. The parameter $C$ controls the fraction of the $K$ participants that update the model in round $t$.

In each round, the $k$-th participant locally takes several steps of SGD on the current model using his entire local training dataset of size $n^k$ (i.e., the globally visible updates are based not on batches but on participants' entire datasets). In the algorithm description, $n$ corresponds to the total size of the training data, i.e., the sum of all $n^k$. Each participant submits the resulting model to the server, which computes a weighted average of the models. The server evaluates the resulting joint model on a held-out dataset and stops training when performance stops improving.

The convergence rate of both collaborative learning approaches heavily

**Algorithm 2:** Federated learning with model averaging

**Server executes:**
  Initialize $\theta_0$
  $m \leftarrow max(C \cdot K, 1)$
  **for** $t = 1$ *to* $T$ **do**
    $S_t \leftarrow$ (random set of m clients)
    **for** *each client* $k \in S_t$ **do**
      $\theta_t^k \leftarrow$**ClientUpdate**$(\theta_{t-1})$
    $\theta_t \leftarrow \sum_k \frac{n^k}{n} \theta_t^k$ ▷ averaging local models

**ClientUpdate**$(\theta)$:
  **for** *each local iteration* **do**
    **for** *each batch* $B$ *in client's split* **do**
      $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(B; \theta)$
  **return** local model $\theta$

depends on the specific learning task and the hyper-parameters (e.g., number of participants, batch size, etc.).

# Chapter 3

# Related Work

In this chapter, we discuss prior work on privacy-friendly data processing, focusing on private statistics and privacy in machine learning.

## 3.1 Private Statistics

This section reviews prior work on privacy-preserving techniques applied to data aggregation, as well as efficient data structures for succinct representation.

### 3.1.1 Privacy-preserving aggregation

Private aggregation schemes like ANONIZE [97] and PrivStats [173] rely on anonymizing networks, such as Tor [65], to protect privacy against network adversaries. Unfortunately, Tor-based schemes are vulnerable to traffic-analysis attacks [155]. Dining Cryptographers anonymizing networks, or DC-nets, are instead used in [58, 60, 61]. However, DC-nets require expensive operations at the servers side.

Other schemes [120, 39, 107, 59] provide strong privacy guarantees via secret-sharing based methods. In particular, Kursawe et al. [120] introduce a few cryptographic constructions to aggregate energy consumptions in the context of smart metering, relying on Diffie-Hellman, bilinear maps, and a *"low overhead"* protocol where meters' encryption keys sum up to zero. Our schemes for the private recommender system and location prediction – presented in Chapter 4 – rely on a protocol inspired by [120]'s "low overhead"

protocol, but perform private aggregation using succinct data representation rather than the raw inputs. Using Count-Min Sketch [56], we reduce computation and communication overhead incurred by each user from linear to logarithmic in the size of the input. We also show how to recover from node failures, i.e., in our schemes, the aggregator can still retrieve the statistics (and train models) even when a subset of users go offline or fail to report data.

Then, Castelluccia et al. [39] propose a new homomorphic encryption to allow intermediate wireless sensor nodes to aggregate encrypted data gathered from other nodes. Jawurek et al. [107] propose a privacy-friendly aggregation scheme with robustness against missing user inputs, by including additional authorities that facilitate the protocol but do not learn any secrets or inputs. However, at least one of the authorities has to be honest, i.e., if all collude, the protocol does not provide any privacy guarantee. Corrigan-Gibbs et al. [59] propose Prio, in which users distribute their trust across non-colluding servers. Their proposal relies on secret-shared non-interactive proofs (SNIPs) to allow user inputs to be validated.

A combination of homomorphic encryption and differential privacy has been explored by Chen et al. [48], allowing third parties to gather web analytics. Users encrypt their data using the data aggregator public key and send them to a proxy, who adds noise to the cipher-texts and forwards the results to the data aggregator. The latter computes the aggregates after decrypting each individual contribution. However, this scheme introduces a large overhead both in terms of communication (one KB per single bit of user data) and computation (one public key operation per single bit). In the same line of work, Akkus et al. [7] propose a system providing differential privacy guarantees. Their scheme scales better than [48] as it requires users to encrypt fewer bits per query, but still relies on expensive public-key crypto operations. In [47], the authors propose a scheme based on a similar trust model as [48] but with an enhanced scalability by using simple exclusive-or (XOR) operations rather than public key operations. However, their proposal still relies on honest-but-

curious servers that do not collude with each other. Shi et al. [188] combine private aggregation with differential privacy supporting the aggregation of encrypted perturbed readings reported by the meters. Individual amounts of random noise cancel each other out during aggregation, except for a specific amount that guarantees computational differential privacy. Their protocol is also so that encryption keys sum up to zero, but requires solving a discrete logarithm and the presence of a trusted dealer. Chan et al. [42] provides fault tolerance by extending [188]'s protocol, however, with a poly-logarithmic penalty.

Erlingsson et al. [80] introduce RAPPOR, which enables the collection of browser statistics on values and strings provided by a large number of clients (e.g. homepage settings, running processes, etc.), including categories, frequencies, and histograms. RAPPOR supports privacy-preserving data-collection mechanism by relying on randomized responses via input perturbation, aiming to guarantee local differential privacy for individual reports. Then, Fanti et al. [82] extend RAPPOR by providing a decoding mechanism that enables the estimation of values which are not part of the initial dictionary. However, RAPPOR requires millions of users in order to obtain approximate answers to queries.

In a different line of work, Burkhart et al. [35] rely on optimized secure multiparty computation (MPC) to allow secure summation tasks, although incurring high bandwidth and computational costs.

Finally, other proposals [4, 106, 77, 30] leverage pairwise additive masking with stream ciphers. Among these works, Elahi et al. [77] present a protocol for privately computing mean statistics on Tor traffic. They introduce two ad-hoc protocols relying, respectively, on secret sharing and distributed decryption. By contrast, our application for gathering private statistics for Tor (presented in Section 4.3) enables the computation of the median statistics on traffic generated by Tor hidden services – which constituted an open problem [88] – by relying on additively homomorphic encryption and differential privacy.

### 3.1.2 Privacy and succinct data representation

Closely related to our contributions presented in Chapter 4 is prior work on privacy-preserving data structures. Bianchi et al [26] formally investigate the privacy guarantees provided by Bloom filters [28]. In particular, they demonstrate that, depending on the Bloom filter parameters, the same level of plausible deniability protection cannot be guaranteed for all elements in the filter. Mir et al. [151] present an efficient scheme guaranteeing differential privacy of data analyses (even when the internal memory of the algorithm may be compromised), using a data structure similar to the Count-Min Sketch to estimate heavy hitters. Work in [100, 41] address the problem of finding heavy hitters' histograms while preserving privacy using a differentially private protocol. Then, [18] addresses the case where individual users randomize their own data and then send differentially private reports to an untrusted server handling reports aggregation.

Other proposals combine differential privacy and Count-Min Sketch to obtain aggregate information about vehicle traffic [153] as well as summaries of sparse databases [57]. Ashok et al. [14] present a privacy-preserving protocol for computing the set-union cardinality among several parties using Bloom filters. However, their proposal is insecure, as shown by [200], who also introduces a novel Bloom filter based protocol for set-union and set-intersection cardinality. Lin et al. [130] improve the performance of [158]'s protocol for private proximity testing by reducing the problem to simple equality testing (instead of the more expensive private-preserving threshold set intersection). They use a concise representation of "location tags", by generating, via shingling, concise sketches—in their context, short strings representing the set of broadcast messages received. Recently, Alaggan et al. [9] propose a sanitization mechanism for approximate distinct counting for Wi-Fi analytics based on perturbed Bloom filters. Their work rely on a variant of differential privacy called *Pan-Privacy* [73] to expand the range of applications enabled by privacy-preserving Bloom filters [8, 205].

To the best of our knowledge, our work, presented in Chapter 4, is the first to show how to combine Count-Min Sketch and privacy-friendly data aggregation to build a private estimated model used for recommendations as well as prediction of future locations. Also, our scheme for Tor hidden services statistics, which combines Count Sketch, additively homomorphic threshold decryption, and differential privacy, is the first to tackle the problem of efficiently computing the median statistics.

## 3.2 Privacy in Machine Learning

Over the past few years, privacy in machine learning and data mining has received a lot of attention from the research community. In this section, we review prior work on privacy-preserving mechanisms applied to machine learning along with inference attacks against machine learning.

### 3.2.1 Learning with privacy

Privacy-enhancing tools based on secure multi-party computation (MPC) and homomorphic encryption have been proposed to securely train supervised machine learning models, such as matrix factorization [160], linear classifiers [31, 89], decision trees [32, 131], linear regressors [68], and neural networks [30, 67, 133, 152].

Among these works, MPC has been used to build privacy-preserving neural networks in a distributed fashion. For instance, SecureML [152] starts with the data owners (clients) distributing their private training inputs among two non-colluding servers during the setup phase; the two servers then use MPC to train a global model on the clients' encrypted joint data. Then, Bonawitz et al. [30] use secure multi-party aggregation techniques, tailored for federated learning, to let participants encrypt their updates so that the central parameter server only recovers the sum of the updates. In Chapter 7, we show that inference attacks can be successful even if the adversary only observes aggregated updates.

Due to its generality, differential privacy has served as a building block

in several recent efforts at the intersection of privacy and deep learning [1, 171, 172, 190]. In general, the majority of privacy-preserving learning schemes focus on convex optimization problems [19, 46, 211], whereas, training neural networks typically requires to optimize non-convex objective functions – as with Restricted Boltzmann Machine (RBM) [38] and Variational Autoencoder (VAE) [115] – which is usually done through the application of Stochastic Gradient Descent (SGD) with poor theoretical guarantees. For instance, Wu et al. [211] propose a private technique which runs SGD for convex cases for a constant number of iterations and only adds noise to the final output.

In the non-convex setting, Shokri and Shmatikov [190] support distributed training of deep learning networks in a privacy-preserving way. Specifically, their system relies on the input of independent entities which aim to collaboratively build a machine learning model without sharing their training data. To this end, they selectively share subsets of noisy model parameters during training. However, their approach incurs high levels of privacy loss per entity, i.e., the $\varepsilon$ parameter is in the order of thousands, using the strong composition theorem [71]. Then, Abadi et al. [1] introduce an algorithm for non-convex deep learning classifiers with strong differential privacy guarantees. They present a privacy accounting method, called the moments accountant, which guarantees a tighter bound of the privacy loss for the composition of multiple gaussian mechanisms when compared to the strong composition theorem [71]. In Chapter 5, we rely on the moments accountant to measure privacy loss, but we train generative models and with an improved gradient descent, where the noise is carefully adjusted and injected in each iteration. Other schemes only focus on learning private autoencoders [171] and convolutional deep belief networks [172] by perturbing their loss functions through the functional mechanism [224].

In the distributed setting, recent proposals [144, 85] tackle the problem of training deep learning models with user-level differential privacy guarantees for the tasks of training language models [144] and digits classification [85].

As opposed to record-level differential privacy (as done in [1]), in user-level differential privacy the aim is to hide whether a client participated during decentralized training. However, these approaches require a large number of users (in the order of thousands) for the training to converge and achieve an acceptable trade-off between privacy and model performance.

In [168], Pathak et al. present a differentially private global classifier hosted by a trusted third-party and based on locally trained classifiers held by separate, mutually distrusting parties. Hamm et al. [91] use knowledge transfer to combine a collection of models trained on individual devices into a single model, with differential privacy guarantees. Transfer learning approaches, in combination with improved differentially private techniques tailored for deep learning [1], have also been applied in [167, 166]. These privately train a "student" model by transferring the knowledge of an ensemble of "teachers" trained on the disjoint subsets of training data, through noisy aggregation.

Finally, few proposals attempt to learn flexible representations of personal users' data via adversarial machine learning [109, 139, 76], transfer learning [163] or Denoising Autoencoders [140], aiming to provide only desired features to a third-party analytics service, while protecting against undesired inference of sensitive tasks (e.g., person identification). However, it is not clear whether these proposals can be applied to the federated training of deep learning networks.

### 3.2.2 Private data release

In the private data release problem, a database owner wishes to release a "sanitized" version of a database, which can be used for data-mining and preserves the privacy of the individuals in the database at the same time. To this end, several techniques for data anonymization have been proposed over the years, including k-anonymity [196] and the related mechanisms of l-diversity [137] and t-closeness [128]. These paradigms aim to protect data by generalizing and suppressing certain identifying attributes, but they cannot be applied to high-dimensional datasets [6, 34]. In fact, Brickell et al. [34]

show that modest privacy gains require almost complete destruction of the data-mining utility.

Therefore, rather than pursuing input sanitization, prior work has proposed techniques to produce plausible synthetic records with strong privacy guarantees, e.g., focusing on differentially private release of data [2, 43, 49, 105, 138, 142, 210, 93]. Alas, these can often support only the release of succinct data representations, such as histograms or contingency tables. In particular, Hardt et al. [93] introduce MWEM, a differentially private algorithm producing synthetic datasets supporting any set of linear queries. MWEM relies on the Exponential Mechanism (EM [146]) to select only the queries most informative to the Multiplicative Weights algorithm [92], which iteratively improves an approximate version of the dataset to better resemble the real one. Instead, several mechanisms protect privacy by adding noise directly to a generative model [33, 127, 132, 225]. In Chapter 5, we follow this approach, while, in a first-of-its-kind attempt, focusing on building private generative machine learning models based on neural networks.

Other approaches [27, 179, 178] generate data records first, and then attempt to test their privacy guarantees, i.e., decoupling the generative model from the privacy mechanism. For instance, Bindschaedler et al. [27] rely on graphical probabilistic models to learn a transformation between real data points (seeds) into synthetic ones. Then, a privacy test filters synthetic data points using a plausible deniability criterion. By contrast, in Chapter 5, we attempt to achieve privacy during the training of the model, thus avoiding eventual high sample rejection rates due to privacy tests.

Recent proposals [21, 202] focus on training differentially private Generative Adversarial Networks (GANs) [87]. Beaulieu et al. [21] apply the noisy gradient descent from [1] to train the discriminator of a GAN under differential privacy. The resulting model is then used to generate synthetic subjects based on the population of clinical trial data. Similarly, Triastcyn et al. [202] perturb the output of the discriminator of a GAN by clipping the L2 norm of the

second-to-last layer of the network and then adding Gaussian noise. However, their approach only provides approximate bounds on the expected privacy loss for the generated dataset, thus failing to achieve *strict* differential privacy guarantees. Moreover, differentially private GANs proposals exacerbate the existing weaknesses of GANs, such as mode collapse and training instability, due to the injection of additional noise into the model during training.

Privacy-aware adversarial training of neural networks, seeking to attain an information-theoretically optimal tradeoff between minimizing distortion of useful data and hiding sensitive information, has also been applied in [101, 203]. However, unlike differential privacy, this approach requires knowledge of the underlying statistical distribution of the data.

Prior work has also attempted to combine clustering with deep learning, albeit with no privacy guarantees. Some proposals [227, 103, 216] jointly train an autoencoder neural network with a clustering algorithm, and use the internal representation provided by the autoencoder, i.e., the encoder output, as features for clustering. A different training method is followed in [126, 66, 214], where autoencoders are initially pretrained, and then fine tuned using the cluster assignment loss, while [99, 220] combine clustering with standard convolutional neural networks (CNNs) for representation learning of images.

Finally, differentially private clustering with $k$-means has been addressed in prior work [195], however, aiming to find linearly separable clusters and add noise which is proportional to the data dimension or the $L_1$-norm of data records. Kernel $k$-means clustering with random Fourier features (RFF) has been considered in [53], albeit without any privacy guarantee. Our work in Chapter 5 somewhat combines [53] and [29], and applies DP $k$-means on Fourier features to ultimately achieve better accuracy than [29].

### 3.2.3 Membership inference attacks

Membership inference is the problem of deciding, given a data point, whether or not it was included in the training dataset. Prior work demonstrated membership inference from aggregate statistics, e.g., in the context

of genomic studies [98, 17], location time-series [174], or noisy statistics in general [74].

Membership inference against black-box machine learning models has been studied in [189, 222, 134]. Shokri et al. [189] demonstrate membership inference against black-box supervised models. Their approach exploits differences in the model's response to inputs that were or were not seen during training. For each class of the targeted black-box model, they train a *shadow model*, with the same machine learning technique. By contrast, our membership attacks, presented in Chapter 6, target generative models and rely on GANs to provide a general framework for measuring the information leakage. Membership inference on generative models is much more challenging than on discriminative models: in the former, the attacker cannot exploit confidence values on inputs belonging to the same classes, and therefore it is more difficult to detect overfitting and mount the attack. In fact, detecting overfitting in generative models is regarded as one of the most important research problems in machine learning [212].

Then, Long et al. [134] and Yeom et al. [222] study the relationship between overfitting and information leakage. In particular, Yeom et al. [222] assume that the adversary knows the distribution from which the training set was drawn and its size, and that the adversary colludes with the training algorithm. Their attacks are close in performance to Shokri et al.'s [189], and show that, besides overfitting, the influence of target attributes on model's outputs also correlates with successful attacks.

Truex et al. [204] extend [189] to a more general setting and show how membership inference attacks are data-driven and largely transferable. They also show that an adversary who participates in collaborative learning, with access to individual model updates from all other honest participants, can boost membership inference performance compared to a centralized model.

To the best of our knowledge, membership inference in generative models has not been studied before.

### 3.2.4 Other attacks on machine learning models

Attacks targeting distributed recommender systems [36] have focused on inferring which inputs cause output changes by looking at model's temporal patterns.

Model inversion techniques infer class features and/or construct class representatives if the adversary has black-box [83, 84] or white-box [15] access to a classifier model.

In particular, Fredrikson et al. [84] show how an attacker can rely on outputs from a classifier to infer sensitive features used as inputs to the model itself: given the model and some demographic information about a patient whose records are used for training, an attacker might predict sensitive attributes of the patient.

These techniques are sometimes described as violating privacy of the training data, even though the inferred features characterize an entire class and not specifically the training data, except in the cases of pathological overfitting where the training sample constitutes the entire membership of the class.

Then, Hitaj et al. [96] show that a participant in collaborative deep learning can use GANs to construct class representatives. This technique has been evaluated only on models where all members of the same class are visually similar (handwritten digits and faces). Thus, there is no evidence that it produces actual training images or can distinguish a training image and another image from the same class. In fact, class members produced by model inversion and GANs are similar to the training inputs only if all members of the class are similar, as is the case for MNIST (the dataset of handwritten digit used in [96]) and facial recognition. This does not violate privacy of the training data; it simply shows that machine learning works as it should. A trained classifier reveals the input features characteristic of each class, thus enabling the adversary to sample from the class population. For instance, Figure 3.1 shows GAN-constructed images for the gender classification task on the LFW dataset, which is one of our experiments (see Section 7.2). These images show

**Figure 3.1:** Samples from a GAN attack on a gender classification model where the class is "female".

a generic female face, but there is no way to tell from them whether an image of a specific female was used in training or not.

Therefore, the informal property violated by the attacks of [83, 84, 15, 96] is: "a classifier should prevent users from generating an input that belongs to a particular class or even learning what such an input looks like". However, it is not clear why this property is desirable or whether it is achievable.

Aono et al. [11] show that, in the collaborative deep learning protocol of [190], an honest-but-curious server can partially recover participants' data points from the shared gradient updates in a greatly simplified setting where the batch consists of a single data point. Furthermore, the technique is evaluated only on MNIST where, as mentioned above, all class members are visually similar. Again, it is not clear if the technique can distinguish a training image and another image from the same MNIST class.

Song et al. [193] engineer a machine learning model that memorizes the training data, which can then be extracted with black-box access to the model, without affecting the accuracy of the model on its primary task. Carlini et al. [37] show that deep learning-based generative sequence models trained on text data can unintentionally memorize specific training inputs, which can then be extracted with black-box access. Even though the models are trained on text, extraction is demonstrated only for sequences of digits (artificially introduced into the text), which are not affected by the relative word frequencies in the language model.

Finally, other attacks focus on model stealing [201, 209, 162]. A stolen model can reveal training data points that are explicitly incorporated or otherwise memorized in the model.

## Chapter 4

# Efficient Privacy-Preserving Computation of Statistics

As discussed in Section 3.1, prior work has proposed a few cryptographic tools for privacy-enhanced computation that could be used for private collection of statistics. For instance, by relying on homomorphic encryption and/or secret sharing, an untrusted aggregator can receive encrypted readings from users and only decrypt their sum [120, 77, 188]. However, these require users to perform a number of cryptographic operations, and transmit a number of ciphertexts, linear in the size of their inputs, which makes it impractical for some scenarios in which inputs to be aggregated are quite large. This motivates the need for efficient and scalable techniques allowing providers to privately gather statistics, and to use such statistics to train models and facilitate predictions.

In this chapter, we present efficient mechanisms that combine traditional linear aggregation with succinct data structures, for efficiency, and, when needed, differential privacy to limit information leakage. Specifically, we show how to use our techniques to instantiate real-world privacy-friendly systems, supporting recommendations for media streaming services, prediction of user locations, and computation of median statistics for Tor hidden services.

## 4.1 Private Recommender Systems For Streaming Services

Media streaming services are becoming increasingly popular as numerous dedicated providers (e.g., Netflix, Amazon, Hulu) as well as "traditional" broadcasting services (e.g., BBC, CNN, Al-Jazeera) offer digital access to TV shows, movies, documentaries, and news. One of the providers' goals is often continuous user engagement, thus, new content should periodically be suggested to users based on their interests. These recommendations are usually provided by means of *recommender systems* [95, 5] like ItemKNN (cf. Section 2.3.1), which typically require the full availability of users' ratings, whereas, we focus on a model where a provider like the BBC provides recommendations to its users, e.g., on iPlayer, *without tracking* their preferences and viewings.

### 4.1.1 Overview

We present a novel privacy-friendly recommender system where the ItemKNN algorithm is trained using only aggregate statistics. Aiming to build a global matrix of co-views (i.e., pairs of programs watched by the same user) in a privacy-preserving way, we rely on (i) private data aggregation based on secret sharing (inspired by the "low overhead protocol" in [120]), and (ii) the Count-Min Sketch (see Section 2.2) data structure to reduce the computation/communication overhead, trading off an upper-bounded error with increased efficiency.

Recommendations are derived, based on ItemKNN, as follows: users' interests are modeled as a (symmetric) item-to-item matrix $I = \{0,1\}^{M \times M}$, where $I_{ab}$ is set to 1 if the user has watched both programs $a$ and $b$ and to 0 otherwise. $I_{aa}$ is set to 1 if the user has watched the program $a$. The Cosine Similarity $\{Sim\}_{ab}$ between programs $a$ and $b$ can be computed from item-to-item matrices (see Equation 2.1 in Chapter 2). The Cosine Similarity is then used by each user to derive personalized recommendations as described

in Section 2.3.1.

**System model.** Our system involves a `tally` (e.g., the BBC) and a set of `users`, and no other trusted/semi-trusted authority:

1. `Users`, possibly organized in groups, compute their (secret) blinding factors, based on the public keys of the other `users`, in such a way that they all sum up to zero. They encrypt their local Count-Min Sketch entries (representing their co-view matrix) using these blinding factors, and send the resulting ciphertexts to the `tally`.

2. The `tally` receives the encrypted Count-Min Sketch from each `user`, aggregates the encrypted counts, and decrypts the aggregates. These are broadcast back to the `users`, who use them to recover an estimate of the global similarity matrix and derive personalized ItemKNN-based recommendations.

**Notation.** In the rest of this section, we denote with $N$ the number of `users`, with $M$ the total number of items, and with $L = d \cdot w$ the number of items in a Count-Min Sketch table $\tilde{X}$. Let $\mathbb{G}$ be a cyclic group of prime order $q$ for which the Computational Diffie-Hellman problem (CDH) is hard and $g$ be the generator of the same group. $\mathrm{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$ denotes a cryptographic hash function mapping strings of arbitrary length to integers in $\mathbb{Z}_q$. Finally, "||" denotes the concatenation operator and $a \in_r A$ means that $a$ is sampled at random from $A$. We assume the system runs on input public parameters $\mathbb{G}, g, q$, where $g$ generates a group of order $q$ in $\mathbb{G}$.

### 4.1.2 Protocol

We now present the details of our proposed protocol. Its cryptographic layer is also summarized in Figure 4.1.

**Setup.** Each `user` $\mathcal{U}_i$ ($i \in [1, N]$) generates a private key $x_i \in_r \mathbb{G}$, and computes and publishes public key $y_i = g^{x_i} \bmod q$. Public keys of all `users` are distributed to each other, using a public bulletin board or the `tally` itself.

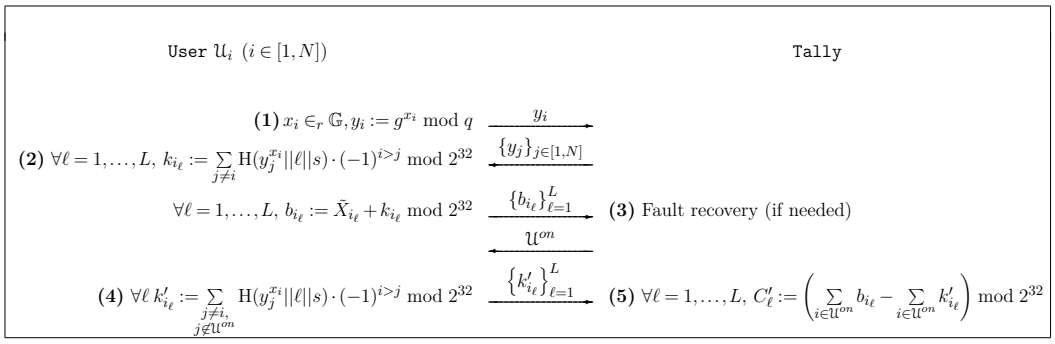



**Figure 4.1:** Cryptographic layer of our private recommender system for online streaming services. At setup **(1)**, `users` compute their secret share and send their public key to the `tally`, who broadcasts them to the other `users`. During the encryption phase **(2)**, each `user` computes the blinding factors, encrypts their Count-Min Sketch and sends it to the `tally`. In case not all `users` have sent the data, the `tally` broadcasts $\mathcal{U}^{on}$, the subset of `users` that did **(3)**. These compute new blinding factors and send them to the `tally` **(4)**. Aggregate sketches are then recovered by the `tally` **(5)**.

As discussed later in this section, `users` might be organized in *groups* in order to facilitate aggregation. To ease presentation, we discuss the protocol steps for a single group of `users`, as combining aggregates from different groups is trivial and can be done, in the clear, by the `tally`.

**Count-Min Sketch construction.** We assume each `user` $\mathcal{U}_i$ holds an input vector of data points $I = \{I_c \in \mathbb{N}, c = 1,\ldots,T\}$, which represents $\mathcal{U}_i$'s co-view matrix (i.e., $T = M \cdot M/2$). First, $\mathcal{U}_i$ initializes a Count-Min Sketch table $\tilde{X}_i$ with all zero entries. In the following, we represent $\mathcal{U}_i$'s Count-Min Sketch table $\tilde{X}_i \in \mathbb{N}^{d \times w}$ as a vector of length $L = d \cdot w$. Then, $\mathcal{U}_i$ encodes $I$ in the Count-Min Sketch using the update procedure described in Section 2.2, where the following pairwise-independent hash function is employed:

$$h(x) = ((ax + b) \bmod p) \bmod w$$

for $a \neq 0, b$ random integers modulo a random prime $p$. At the end of this step, $\mathcal{U}_i$ has built a Count-Min Sketch table $\tilde{X}_i = \{\tilde{X}_{i_\ell}\}_{\ell=1}^{L}$ (with $L = d \cdot w$ as per Definition 7).

**Encryption.** To participate in the privacy-preserving sketch aggregation, each `user` $\mathcal{U}_i$ first needs to generate blinding factors. At round $s$, for each $\ell = 1, \ldots, L$, `user` $\mathcal{U}_i$ computes:

$$k_{i_\ell} = \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathrm{H}(y_j^{x_i} || \ell || s) \cdot (-1)^{i>j} \bmod q$$

where

$$(-1)^{i>j} := \begin{cases} -1 & \text{if } i > j \\ 1 & \text{otherwise} \end{cases}$$

Note that the sum of all $k_{i_\ell}$'s equals to zero:

$$\sum_{i=1}^{N} k_{i_\ell} = \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \mathrm{H}(y_j^{x_i} || \ell || s) \cdot (-1)^{i>j} = 0$$

Then, for each entry $\tilde{X}_{i_\ell}$, $\mathcal{U}_i$ encrypts $\tilde{X}_{i_\ell}$ as $b_{i_\ell} = \tilde{X}_{i_\ell} + k_{i_\ell} \bmod 2^{32}$, as only 32 bits of $b_{i_\ell}$ are enough for our application, and sends the resulting ciphertext to the `tally`.

**Aggregation.** The `tally` receives the ciphertexts from the $N$ `users` and (obliviously) aggregates the sketches. Specifically, for $\ell = 1, \ldots, L$, it computes:

$$C_\ell = \sum_{i=1}^{N} b_{i_\ell} = \sum_{i=1}^{N} k_{i_\ell} + \sum_{i=1}^{N} \tilde{X}_{i_\ell} = \sum_{i=1}^{N} \tilde{X}_{i_\ell} \bmod 2^{32}$$

where $C_\ell$ denotes the $\ell$-th item in the aggregate Count-Min Sketch table. $\{C_\ell\}_{\ell=1}^{L}$, are broadcast back to the `users` (but can obviously be used locally at the `tally` too), who use them to recover an estimate of the global matrix and derive personalized recommendations via the ItemKNN algorithm.

**Fault tolerance.** If, during the aggregation phase, only a subset of `users` report their values $b_{i_\ell}$ to the `tally`, the sum of the $k_{i_\ell}$'s is no longer equal to zero and the aggregate items $C_\ell$ cannot be decrypted. However, it is possible to recover as follows: Let $\mathcal{U}^{on}$ denote the list of `users` who have submitted the data in the aggregation phase. The `tally` sends $\mathcal{U}^{on}$ to each $\mathcal{U}_i \in \mathcal{U}^{on}$. Then, $\mathcal{U}_i$ computes, for each $\ell = 1, \ldots, L$,

$$k'_{i_\ell} = \sum_{\substack{j=1 \\ j \neq i, j \notin \mathcal{U}^{on}}}^{N} \mathrm{H}(y_j^{x_i} || \ell || s) \cdot (-1)^{i>j} \bmod q$$

and sends these values back to the `tally`. Note that $|\mathcal{U}^{on}|$ should be at least 3, otherwise there would be no privacy guarantees for the `users` in $\mathcal{U}^{on}$. Assuming all `users` in $\mathcal{U}^{on}$ submit the values $k'_{i_\ell}$, the `tally` can recover the entries in the aggregate sketches (for `users` in $\mathcal{U}^{on}$) by computing:

$$C'_\ell = \left( \sum_{i \in \mathcal{U}^{on}} b_{i_\ell} - \sum_{i \in \mathcal{U}^{on}} k'_{i_\ell} \right) \bmod 2^{32}$$

**Groups.** Although the protocol can cope with faults, we should nonetheless minimize the probability of missed contributions. Moreover, as discussed in Section 4.1.4, the protocol's complexity also depends on the number of `users` and, in the case of iPlayer, there can be peaks of hundreds of thousands of users per hour[1]. Consequently, we need to organize `users` into reasonably sized groups. As mentioned earlier, combining aggregates from different groups is straightforward and can be done, in the clear, by the `tally`.

We argue that a good choice is between 100 and 1,000 `users` per group, as also supported by our empirical evaluation presented later. There could be a few different ways to form groups: for instance, the `tally` could group `users` in physical proximity and/or select `users` that are watching/listening a video with at least a couple of minutes left to watch. Also note that `users` not involved in the protocol (or having limited "history") can get recommendations too as the `tally` can still provide them with the global co-view matrix, which, even though it does not include their own contribution, can be used by the ItemKNN algorithm to derive recommendations.

**Security analysis.** The security of our scheme, in the honest-but-curious model, is straightforwardly guaranteed by that of the "low overhead" private aggregation scheme by Kursawe et al. [120], which is secure under the CDH assumption. We modify it to cope with users faults and to aggregate Count-Min Sketch entries, rather than the actual data, and this does not affect the privacy properties of the scheme. In case of passive collusions between `users`, the confidentiality of the data provided by the non-colluding `users` is still preserved.

[1] `http://downloads.bbc.co.uk/mediacentre/iplayer/iplayer-performance-may17.pdf`

Finally, note that malicious active `users` could report fake values in order to invalidate the final aggregation values, however, protocol's integrity could be preserved using verifiable tools such as zero-knowledge proofs and commitments, an extension we leave as part of future work, along with considering a malicious `tally`.

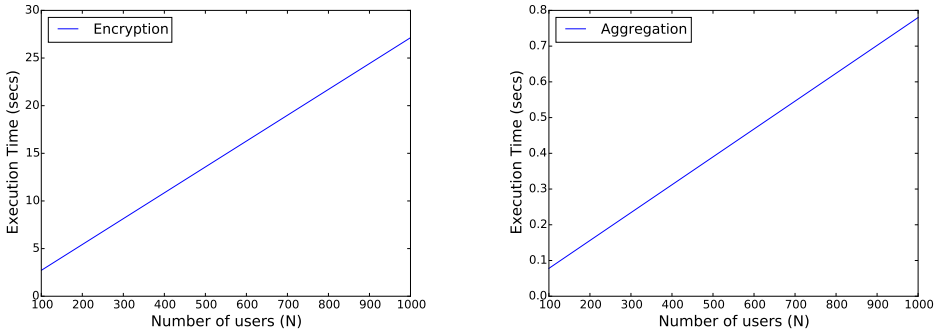
### 4.1.3 Prototype implementation

We have implemented the `tally`'s functionalities as a web application running on the server-side JavaScript environment *Node.js* (or *Node* for short).[2] We also use *Express.js*[3] to organize our application into a Model View Controller (MVC) web architecture and *Socket.io*[4] to set up bidirectional web-socket connections. Integrating our solution is as simple as installing a *Node* module through the Node Package Manager (NPM) and importing it from any web application, thus requiring no familiarity with the inner workings of the cryptographic and aggregation layers.

The module for `user`'s functionalities is modeled as the client-side of the web application and can be run as simple JavaScript code embedded on a HTML page. Therefore, it requires no deployment or installation of any additional software by the `users`, but runs directly in the browser, transparently, when `users` visit `tally`'s website. Our JavaScript implementation is also compatible with smartphone browsers (e.g., the Android version of Chrome), nevertheless, we have also implemented a stand-alone Android application using Apache Cordova.[5]

**Cryptographic operations.** The cryptographic layer of the protocol is also written in JavaScript, using the Ed25519 curve [24] implementation available from *Elliptic.js*,[6] which supports 256-bit points and provides security comparable to a 128-bit security parameter. SHA-256 is used for (cryptographic)

[2] `https://nodejs.org/`
[3] `http://expressjs.com/`
[4] `http://socket.io/`
[5] `https://cordova.apache.org/`
[6] `https://github.com/indutny/elliptic`




**(a)** Client **(b)** Server

**Figure 4.2:** Execution time for increasing number of `users` (with 700 programs).

hashing operations.

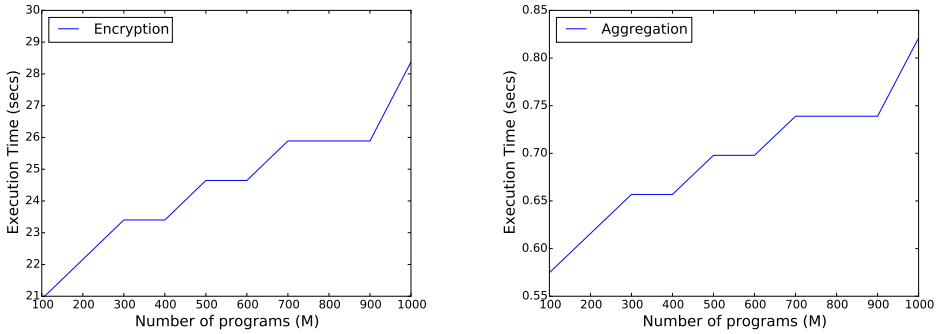
### 4.1.4 Performance evaluation

We now analyze the performance of our system, both analytically (reporting asymptotic complexities) and empirically.

**Asymptotic complexities.** The setup phase carried out by the `user` requires $O(N)$ random group points (where $N$ is the number of total `users`) and $O(N)$ messages need to be sent for all the `users` to distribute the public keys. To generate the blinding factors, each `user` then needs to perform $O(N)$ exponentiations in $\mathbb{G}$ and $O(L \cdot N)$ hashing operations. Count-Min Sketch encryption (at `user`'s side) requires $O(L)$ integer additions in $\mathbb{Z}_q$, one for each of the $L = O(\log(M^2))$ Count-Min Sketch entries, while communication complexity amounts to $O(L)$ 32-bits integers for each `user`. To complete the aggregation, the `tally` computes $O(L \cdot N)$ linear operations.

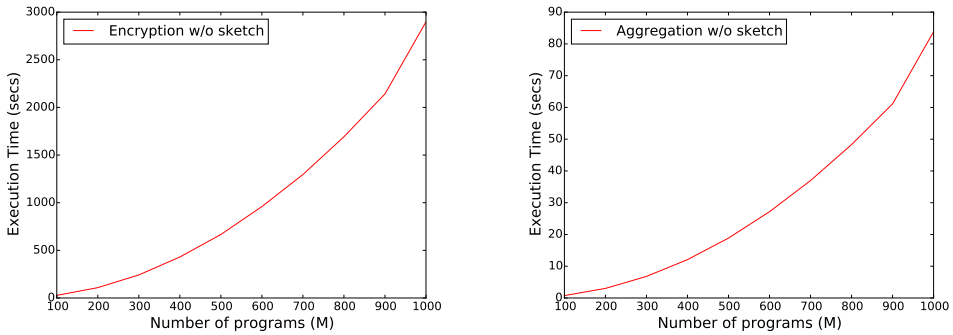
The use of the Count-Min Sketch significantly speeds up the efficiency of the system. In fact, without them, each `user` would need to perform $O(N(M^2))$ hashing operations and send $O(M^2)$ 32-bit integers, while the `tally` would need to compute $O(N(M^2))$ operations.

**Computation overhead.** We have also simulated the execution of our private recommender system and measured execution times (averaged over 100 iterations) for all operations. Simulations have been performed on a machine




**(a)** Client **(b)** Server

**Figure 4.3:** Execution time for increasing number of programs (with 1,000 `users`).




**(a)** Client **(b)** Server

**Figure 4.4:** Execution time for increasing number of programs (with 1,000 `users`) *without* Count-Min Sketch.

running Ubuntu Trusty (Ubuntu 14.04.2 LTS), equipped with a 2.4 GHz CPU i5-520M and 4GB RAM.

In Figure 4.2, we plot running times of protocol's client- and server-side for an increasing number of `users`, fixing the number of programs to 700 (the average number of programs available on iPlayer) and the sketch parameters to $\epsilon = \delta = 0.01$ (see Definition 7). Using this setting, the number of rows $d$ and columns $w$ of the Count-Min Sketch amounts to $d = 18$, $w = 272$ leading to a Count-Min Sketch of size $L = d \cdot w = 18 \cdot 272 =$ 4,896. Running times grow linearly in the number of `users`. As illustrated in Figure 4.2(a), the encryption, performed by each `user` (see step (2) in Figure 4.1), takes 2.7

seconds with 100 `users` and 27 seconds with 1,000 `users`, while Figure 4.2(b) reveals that `tally` completes the aggregation (step (5) in Figure 4.1) in 78ms (resp., 780ms) with 100 (resp., 1,000) `users`.

We then measure the execution time for an increasing number of programs and a fixed number of `users`, i.e., 1,000. Figure 4.3(a) illustrates running times' logarithmic growth for encryption, ranging from 21 seconds with 100 programs to 28 seconds with 1,000 programs. Figure 4.3(b) illustrates `tally`'s execution times for the aggregation, which approximately range from 600ms to 800ms. Note that the "stair" effect of the plots in Figure 4.3 is due to the fact that the Count-Min Sketch size can be the same with close numbers of programs, and thus leads to the same execution times for both `user` and `tally`. On the contrary, the number of users does not affect the size of the Count-Min Sketch (see Figure 4.2)

Without the compression factor of the Count-Min Sketch, the running times for both `user` and `tally` would grow linearly in the size of the co-view matrix (i.e., $M \cdot M/2$), yielding remarkably slower executions. As illustrated in Figure 4.4(a), with 1,000 `users` and 1,000 programs, running time for each `user` amounts to almost 50 minutes instead of 28 seconds using the sketch, whereas, the aggregation at the `tally` completes in almost one and a half minute (versus less than one second using Count-Min Sketch). Finally, execution time of the *ItemKNN* operations carried out at `user`'s side, with 700 programs, amounts to 850ms for each `user`.

**Communication overhead.** In Table 4.1, we report the amount of bytes exchanged between all parties for different number of `users` and Count-Min Sketch sizes (for different choices of sketch parameters $\epsilon$ and $\delta$), fixing the number of programs to 700. Note that, without the compressing factor of the sketch, with 700 programs, each `user` would have to send 960KB instead of 20KB.

**Accuracy estimation.** Finally, we evaluate the accuracy loss due to the use of Count-Min Sketch, specifically, over the most 50 frequent items, using

| #Users | Bytes (Tally to User) | Sketch Size | Bytes (User to Tally) |
|---|---|---|---|
| 100 | 3,200 | 4,896 | 19,584 |
| 200 | 6,400 | 2,448 | 9,792 |
| 300 | 9,600 | 1,638 | 6,552 |
| 400 | 12,800 | 1,224 | 4,896 |
| 500 | 16,000 | 972 | 3,888 |
| 600 | 19,200 | 810 | 3,240 |
| 700 | 22,400 | 702 | 2,808 |
| 800 | 25,600 | 612 | 2,448 |
| 900 | 28,800 | 540 | 2,160 |
| 1000 | 32,000 | 486 | 1,944 |

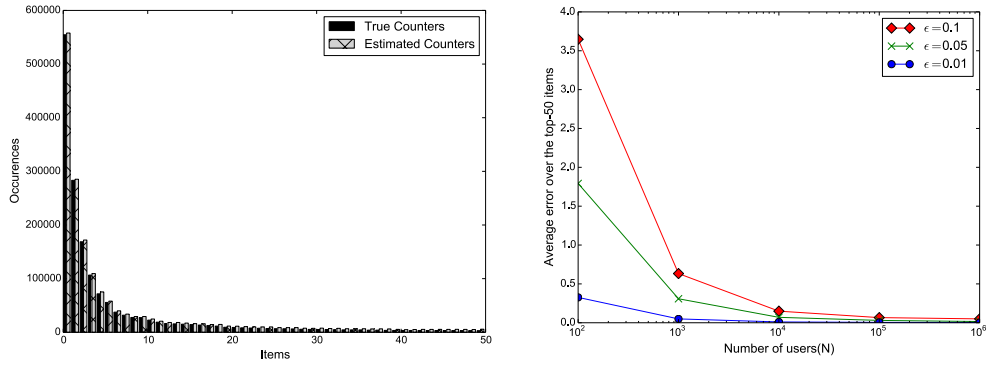
**Table 4.1:** Bytes exchanged by `user` and `tally` for different #`users` and size of the Count-Min Sketch, considering 700 programs.

a synthetic dataset sampled from a zipfian distribution simulating a million `users`. We set the Count-Min Sketch parameters to be $\epsilon = 0.01$ and $\delta = 0.01$ as we have measured an acceptable accuracy loss level introduced by the Count-Min Sketch (see below). Once again, we fix the number of programs to $M = 700$, leading to a Count-Min Sketch of size $L = 4{,}896$. Figure 4.5(a) shows that the Count-Min Sketch estimation over the most 50 frequent items is almost indistinguishable from the true population.

We also plot, in Figure 4.5(b), the average error, defined as $|\hat{c}_i - c_i| / \sum_j |c_j|$, over the most 50 frequent items with an increasing number of `users`, while fixing $M = 700$, $\delta = 0.01$ (yielding a total number of items to update on the Count-Min Sketch of $T = M \cdot M/2 = 245{,}000$) and three choices of the $\epsilon$ parameter, i.e., $0.01, 0.05$, and $0.1$. The average error decreases with more `users` and smaller values of $\epsilon$. Standard deviation values are infinitesimal, thus, we do not include them in the plot as they would not be visible.

## 4.2 Private Aggregate Location Prediction

In this section, we instantiate a mobile application enabling `users` to report, to a service provider (`tally`), their locations over time. `Users`' privacy




**(a)** True vs estimated counters **(b)** Average error for different values of $\epsilon$

**Figure 4.5:** Visualizing the accuracy of the Count-Min Sketch for the most 50 frequent items (with 700 programs and sketch size 4,896).

is protected as only aggregate (over many `users`) location statistics are disclosed. We then show how these statistics can be used to train a model and predict future movements, and support private computation and prediction of "heat maps" relying on the aggregate counts of people in a given area over a period of time. Location heat maps are useful in many different applications as they provide a graphical representation of congestion, using different colors to represent the "popularity" of geographical areas.

**System model.** We operate in the same model as our privacy-friendly recommender system (see Section 4.1.2), involving a `tally` that privately aggregates location statistics contributed from a set of `users`, and re-use the same cryptographic layer. Once again, we support efficient computation of private statistics using (i) Count-Min Sketch's succinct data representation and (ii) privacy-preserving aggregation with `users`' blinding factors summing up to zero.

**Overview.** We assume a 2-D space territory $\mathcal{R}$ is partitioned into a grid of $|S| = p \times p$ cells ($S = \{S[1,1], S[1,2], \ldots, S[p,p]\}$), and $t$ finite intervals (time slots) $[t_{j-1}, t_j]$, where $j \in \mathbb{N}^+$. Let $S_i^{(t_j)}$ be the grid containing, for each cell, the number of times the `user` $\mathcal{U}_i$ has logged her position (using a GPS mea-

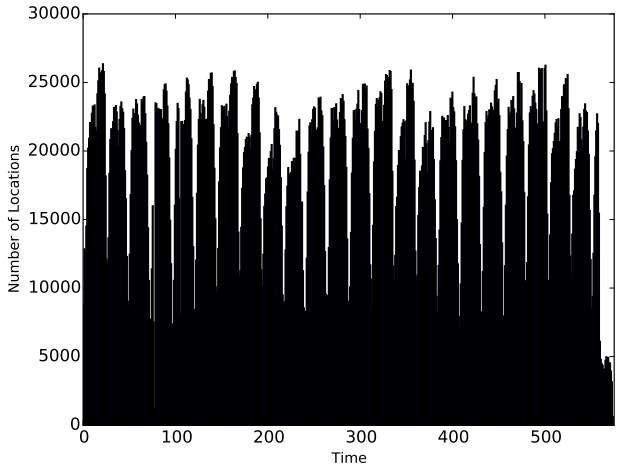



**Figure 4.6:** Number of taxi locations over time.

surement) within that particular cell over $t \in [t_{j-1}, t_j]$. `User` $\mathcal{U}_i$, for each time slot $[t_{j-1}, t_j]$, builds the grid $S_i^{(t_j)}$ with locations logged over time, maps the grid into a Count-Min Sketch, and sends the encrypted sketch to the `tally`. This aggregates and decrypts them, reconstructing the grid containing the (estimated) aggregate locations.

The location statistics can be used to display 'heat maps" (e.g., a graphical representation of congestion), or to perform time-series based prediction over a sequence of heat maps. Using an Exponential Weighted Moving Average (EWMA) model (cf. Section 2.3.2), we can predict the future popularity of a cell, by relying on the past (approximated) observations for that cell. Other machine learning techniques, e.g., Multivariate Support Vector Machines or Logistic Regression, could also be used for the prediction, but we consider it to be beyond the scope of this work to investigate new predictors.

**The San Francisco Cabs dataset.** To evaluate the feasibility of our intuition, we use a publicly available dataset containing mobility traces of San Francisco taxi cabs.[7] The dataset contains 11 million GPS coordinates, generated by 536 taxis over almost a month in May 2008. We group the taxi locations in time slots of one hour, leading to a total of 575 epochs. Figure 4.6 shows the presence of weekly and daily patterns in the number of taxi loca-

[7] `http://cabspotting.org/`

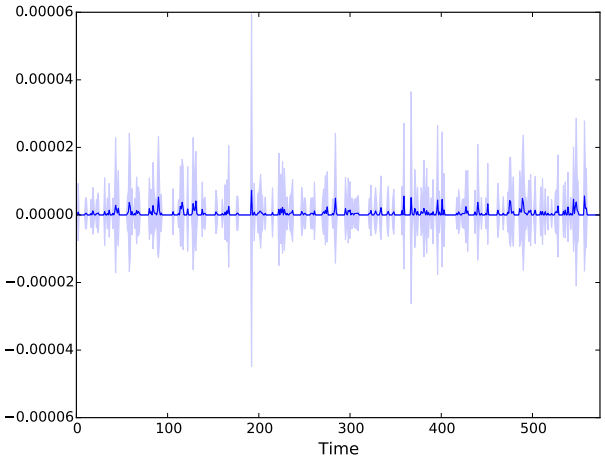



**Figure 4.7:** Average error introduced by the Count-Min Sketch on the aggregate statistics for the top-100 locations.

tions over time (i.e. hourly time slots) and peaks of roughly 25,000 total hourly contributions.

**Succinct data representation.** We investigate whether succinct data representation could be applied to the problem of collecting location statistics, and measure the accuracy loss introduced by the Count-Min Sketch's compact representation. In Figure 4.7, we plot the average error defined as $|\hat{c}_i - c_i| / \sum_j |c_j|$ and the relative standard deviation over the most 100 popular cells for each time slot, while fixing $\epsilon = \delta = 0.01$ and the total number of cells to $|S| = 100 \times 100$ (yielding a Count-Min Sketch of size $L = 3,808$). Observe that the average error is infinitesimal for every time slots.

**Heat map prediction.** Next, we focus on predicting future heat maps using the EWMA algorithm introduced in Section 2.3.2. We start by evaluating the accuracy of EWMA-based prediction relying on the aggregates collected *without using the Count-Min Sketch*. We perform the prediction over a subset of 12 consecutive epochs having the maximum number of reported locations, giving the past 24 hours observations as input to the EWMA algorithm. Figure 4.8 plots the Mean Absolute Error (MAE) in the prediction compared to the ground truth over the most 100 popular cells, considering different values of $\alpha$, i.e., EWMA's smoothing coefficient (cf. Section 2.3.2). The plot shows

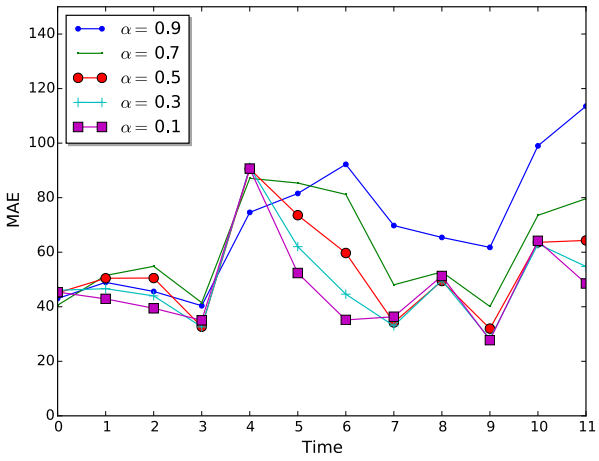



**Figure 4.8:** Mean absolute error in the prediction for different values of prediction algorithm's parameter $\alpha$.

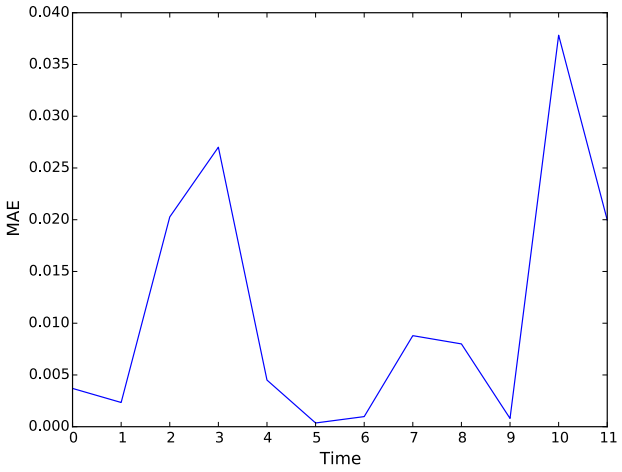



**Figure 4.9:** Mean absolute error introduced by the Count-Min Sketch on the prediction accuracy.

that, in almost all slots, lower values of $\alpha$ lead to more accurate results.

We then perform the prediction over the approximate heat maps, i.e., *using the sketches*. We focus on the same time slot, and fix $\alpha = 0.1$. Figure 4.9 shows the error introduced by the Count-Min Sketch in the prediction, for each time slot considered, with respect to the prediction based on the *"real"* heat maps. We observe that this error, while fluctuating, is appreciably low for every prediction, thus confirming the feasibility of our techniques for the problem of privately predicting future heat maps.

Once again, we have implemented our techniques in JavaScript, with the

server-side running as a *Node* module, and client-side running as an Android application built using Apache Cordova.

## 4.3 Tor Hidden Services Statistics

The privacy-preserving collection of statistics using efficient data structures, seeking a trade-off between accuracy and efficiency, has also interesting applications in non-user facing settings such as collecting network statistics from servers or routers. In this section, we present a novel mechanism geared to privately gather statistics in the context of the Tor anonymity network [65]. The Tor project has recently received funding to improve monitoring of load and usage of Tor hidden services.[8] This motivates them to extract aggregate statistics about the number of hidden service descriptors from multiple Hidden Service Directory authorities. In order to ensure robustness, the Tor project has determined that the median – rather than the mean – of these volumes should be calculated, which is beyond privacy-friendly statistics approaches like Privex [77].

In this section, we first describe the protocol for estimating median statistics using Count Sketch, then, we present the design and deployment of its prototype implementation, along with its performance evaluation.

### 4.3.1 Private median estimation using Count Sketch

We rely on the Count Sketch [44] data structure, which closely resembles Count-Min Sketch, used in Sections 4.1–4.2. Recall from Section 2.2 that building a Count Sketch follows the same process as a Count-Min Sketch, thus leading to a $d \cdot w$ table of positive integer values, whereas, the estimation of an item's frequency is slightly different: for each row, $d_i$, a hash function is applied to the item leading to a column $w_j$. An unbiased estimator of the frequency of the item is the value at this position minus the value at an adjacent position – and the median of those estimators is the final estimated frequency. What is key to the success of our techniques is that the estimate of the frequency of

[8] `https://www.torproject.org/docs/hidden-services.html.en`

specific values, as well as sets of values, is a simple linear sum of Count Sketch entries; computing it does not require non-linear (e.g., *min*) operations as for the Count-Min Sketch.

For this application, we build on privacy-preserving data aggregation based on threshold public-key encryption, specifically, an Additively Homomorphic Elliptic-Curve variant of El Gamal (AH-ECC) [22], summarized below. This allows us to seamlessly tolerate missing contributions – following an approached first proposed by Jawurek et al. [107].

**AH-ECC** consists of the following three algorithms (using a multiplicative notation):

1. *KeyGen*$(1^\tau)$: Given a security parameter $\tau$, choose an elliptic curve $E$ and $(g_1, g_2)$ public generators on $E$, generating a group of order $q$. Choose a random private key $x \in \mathbb{Z}_q$, define the public key as $pk = {g_1}^x$, and output public parameters $(E, g_1, g_2, pk)$ and private key $x$.
2. *Encrypt*$(m, pk)$: The message $m$ is encrypted by computing two elliptic curve points as $(A, B) := ({g_1}^r, pk^r {g_2}^m)$, where $r \in \mathbb{Z}_q$ is selected at random. The ciphertext is thus the tuple of points $(A, B)$.
3. *Decrypt*$(A, B, x)$: Decryption is performed by computing the element $BA^{-x} = {g_2}^m$. We can achieve constant time decryption by pre-computing a table of discrete logarithms which is then used to recover $m$ from ${g_2}^m$ (this solution is practical for small values of $m$).

AH-ECC is additively homomorphic since an element-wise multiplication of ciphertexts yields an encryption of their sum.

**Setup.** Our system relies on a set of authorities that can jointly decrypt a ciphertext from the AH-ECC additively homomorphic public-key cryptosystem. During setup, each authority generates their public and private key and a group public key is computed by multiplying all the authorities' public keys. Note that we operate in a distributed system setting (i.e., the Tor network), therefore, similar to PrivEx [77], one can easily instantiate decryption authorities.

**Protocol.** Using Count Sketch, we can collect a number of private readings from Hidden Service Directories (HSDir), and compute an approximation of the median. Each HSDir builds a Count Sketch, inserts its private values into it, encrypts it, and sends it to the authorities. These aggregate all sketches by homomorphically adding them element-wise, yielding an encrypted sketch summarizing the set of all HSDir values.

Once the authorities have computed the aggregate sketch, an interactive divide-and-conquer algorithm is applied to estimate the median given the range of its possible values is known. At each iteration, the number of sample values in the range is known, starting with the full range and all values received. The range is then halved and the sum of all elements falling in the first half of the range is jointly decrypted. If the sum falls within first half of the range it is retained for the next iteration, otherwise the second half of the range is considered at the next iteration. The process stops once the range is a single element. Following the master theorem [55], we know that this process converges in $O(\log n)$ steps, for $n$ elements in the domain of the values/median. Due to frequency estimations for the ranges using Count Sketches that provide noisy estimates, we expect this median to be close, but possibly not exactly the same as the true sample median, depending on the Count Sketch parameters $\delta$ and $\epsilon$.

**Output privacy.** Note that this process is not "perfectly" private in a traditional secure computation setting, as the volume of reported values falling within the intermediate ranges considered is leaked. This may be dealt with in two ways: (1) the leakage may be considered acceptable and the algorithm run as described, or (2) the technique can be enhanced to provide differential privacy by adding noise to each intermediate value.

**Differentially private estimates.** The sensitivity [72] of the estimates in any range of values using the Count Sketch is at most $d$, since each HSDir contribution increases by at most 1 in at most $d$ values into the $d \cdot w$ Count Sketch table. Therefore, we can achieve $\epsilon$-differential privacy if we add, to

each decrypted value, noise from a Laplace distribution with mean zero and variance $\xi \cdot d/\epsilon$, where $\xi$ is the number of decrypted intermediate results and $\epsilon$ the differential privacy parameter. However, doing so may result in the divide-and-conquer algorithm mis-estimating the range in which the median lies, and results in further mistakes in the final median estimate. As discussed in Section 2.1.3, although we use $\epsilon$ to denote a parameter for both Count Sketch and differential privacy, it is clear from the context which one it relates to.

### 4.3.2 Implementation and evaluation

We implement and evaluate the proposed scheme aiming to: (i) estimate the trade-off between size of the sketch and the accuracy of the median computation, (ii) evaluate the cost of cryptographic computation and communication overheads, and (iii) assess the trade-off between the accuracy of the median and the quality of protection that may be achieved through the differentially private mechanism.

For our evaluation, we instantiate AH-ECC using the NIST-P224 curve as provided by the OpenSSL library and its optimizations by Käsper [113]. Our implementation of the cryptographic core of the private median scheme amounts to 300 lines of Python code using the *petlib* OpenSSL wrapper,[9] and another 350 lines of Python include unit tests and measurement code. All experiments have been performed on a machine running Ubuntu Trusty (Ubuntu 14.04.2 LTS), equipped with a 2.4 GHz CPU i5-520M and 4GB RAM. Our Python implementation is easily pluggable as part of the Tor infrastructure and does not require changes within the Tor (C-based) core functionalities.

We first illustrate the performance and accuracy of estimating the median using this technique with both sketch parameters $\epsilon$ and $\delta$ equal to either 0.25 or 0.05 against the London Atlas Dataset[10] in Table 4.2. The error rate is computed as the absolute value of difference between the estimated and true

[9] `https://github.com/gdanezis/petlib`

[10] `http://data.london.gov.uk/dataset/ward-profiles-and-atlas`

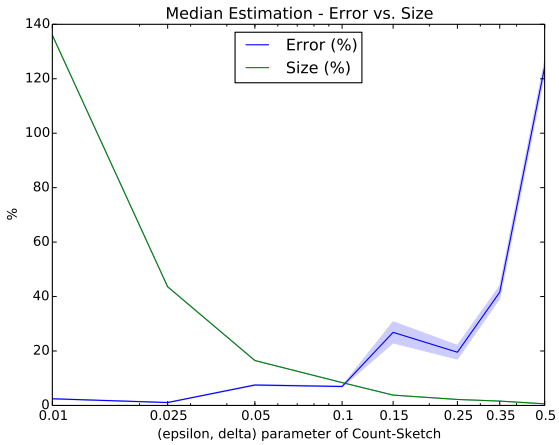



**Figure 4.10:** Count Sketch size versus estimation quality.

median divided by the true median.

Further results are presented on an experimental setup that uses as a reference problem the median estimation in a set of 1,200 sample values, drawn from a mixture distribution: 1,000 values from a Normal distribution with mean 300 and variance 25, and 200 values drawn from a Normal distribution with mean 500 and variance 200. This reference problem closely matches the settings of the Tor project both in terms of the range of vales (assumed to be within $[0, 1000]$) and the number of samples [77].

**Quality vs. Size.** Figure 4.10 illustrates the trade-off between the quality of the estimation of the median algorithm and the size overhead of the Count Sketch. The size overhead (green slim line) is computed as the number of encrypted elements in the sketch as compared with the number of elements in the range of the median (1,000 for our reference problem). The estimation accuracy (blue broader line) is represented as the fraction of the absolute deviation of the estimate from the real value over the real sample median (light blue region represents the standard deviation of the mean over 40 experiments for each datapoint). Thus both qualities can be represented as percentages.

The trade off between the size of the sketch and the accuracy of the estimate is evident: as the sketch size reaches a smaller fraction of the total possible number of values, the error becomes larger than the range of the

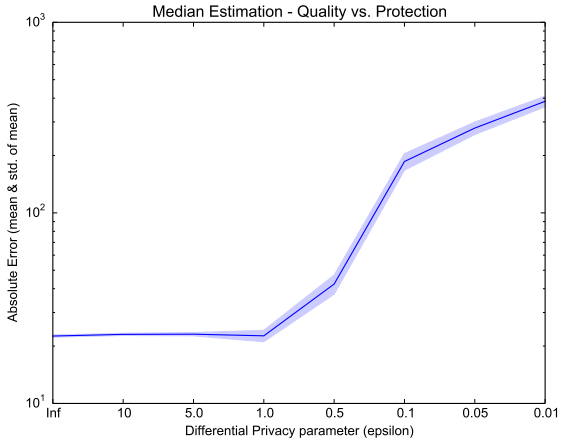



**Figure 4.11:** Quality versus differential privacy protection.

median. Thus, Count Sketch with parameters $\epsilon, \delta < 0.025$ are unnecessary, since they do not lead to a reduction of the information that needs to be transmitted from each client to the authorities; conversely, for $0.15 < \epsilon, \delta$ the estimate of the median deviates by more than 20% of its true value making it highly unreliable.

For all subsequent experiments, we consider a Count Sketch with values $\epsilon = \delta = 0.05$, leading to $d = 3$ and $w = 55$. As outlined in Figure 4.10, this represents a good trade-off between the size of the Count Sketch (16.5% of transmitting all values) and the error.

**True size and performance.** When implemented using NIST-P224 curves, the reference Count Sketch may be serialized in 10,898 bytes. Each Count Sketch takes 0.001 sec to encrypt at each HSDir, and it takes 1.456 seconds to aggregate 1,200 sketches at each authority (0.001 sec per sketch). As expected, from the range of the reference problem, 10 decryption iterations are sufficient to converge to the median (therefore $\xi = 10$). The number of homomorphic additions for each decryption round is linear in the range of the median and their total computational cost is the same order of magnitude as a full Count Sketch encryption. It is clear from these figures that the computational overhead of the proposed technique is eminently practical, and the bandwidth overhead acceptable.

**Quality vs. Differential Privacy protection.** Figure 4.11 illustrates the trade-off between the quality of the median estimation and the quality of differential privacy protection. The x-axis represents the $\epsilon$ parameter of the differentially private system, and the y-axis the absolute error between the estimate and the true sample median. Differential privacy with parameter $\epsilon = 0.5$ can be provided without significantly affecting the quality of the median estimate. However, for $\epsilon < 0.5$ the volume of the error grows exponentially (note the log scale of the x-axis). While the exact value of a meaningful $\epsilon$ parameter is often debated in the literature, we conclude that the mechanism only provides a limited degree of protection, and no ability to readily tune up protection: utility degrades very rapidly as the security parameter $\epsilon$ decreases.

| | Median ($\epsilon, \delta = 0.25$) | Error (%) | Median ($\epsilon, \delta = 0.05$) | Error (%) | Truth |
|---|---|---|---|---|---|
| Population - 2015 | 15143.2 | 11.3 | 13215.4 | 2.8 | 13600.0 |
| Children aged 0-15 - 2015 | 2970.8 | 12.1 | 2627.6 | 0.8 | 2650.0 |
| Working-age (16-64) - 2015 | 9592.0 | 2.0 | 8843.2 | 5.9 | 9400.0 |
| Older people aged 65+ - 2015 | 1284.6 | 11.4 | 1345.0 | 7.2 | 1450.0 |
| % All Children aged 0-15 - 2015 | 21.9 | 10.7 | 20.1 | 1.3 | 19.8 |
| % All Working-age (16-64) - 2015 | 70.7 | 5.0 | 68.8 | 2.2 | 67.3 |
| % All Older people aged 65+ - 2015 | 15.2 | 37.1 | 12.0 | 7.8 | 11.1 |
| Mean Age - 2013 | 38.6 | 8.8 | 36.9 | 3.8 | 35.5 |
| Median Age - 2013 | 37.7 | 10.8 | 35.7 | 5.1 | 34.0 |
| Population density (persons per sq km) - 2013 | 10231.3 | 44.8 | 5792.9 | 18.0 | 7067.0 |
| % BAME - 2011 | 45.6 | 26.3 | 35.7 | 1.0 | 36.1 |
| % Not Born in UK - 2011 | 40.1 | 7.6 | 40.1 | 7.6 | 37.3 |
| Male life expectancy -2009-13 | 84.1 | 5.7 | 79.6 | 0.0 | 79.6 |
| Female life expectancy -2009-13 | 87.0 | 3.5 | 84.9 | 0.9 | 84.1 |
| In employment (16-64) - 2011 | 6532.8 | 7.0 | 5843.7 | 4.2 | 6103.0 |
| Employment rate (16-64) - 2011 | 68.5 | 2.0 | 70.8 | 1.3 | 69.9 |
| Number of properties sold - 2013 | 169.3 | 1.4 | 149.8 | 10.3 | 167.0 |
| Number of Household spaces - 2011 | 5619.1 | 5.4 | 5025.9 | 5.7 | 5332.0 |
| % detached houses - 2011 | 2.4 | 44.7 | 1.6 | 62.2 | 4.3 |
| % semi-detached houses - 2011 | 29.0 | 70.6 | 16.7 | 1.5 | 17.0 |
| % terraced houses - 2011 | 29.4 | 39.8 | 21.1 | 0.6 | 21.0 |
| % Flat, maisonette or apartment - 2011 | 53.1 | 15.1 | 49.7 | 7.9 | 46.1 |
| % Households Social Rented - 2011 | 26.0 | 27.5 | 19.9 | 2.4 | 20.4 |
| % dwellings in council tax bands A or B - 2011 | 21.2 | 79.9 | 10.4 | 12.2 | 11.8 |
| % dwellings in council tax bands C, D or E - 2011 | 63.7 | 7.5 | 71.6 | 3.9 | 68.9 |
| % dwellings in council tax bands F, G or H - 2011 | 0.3 | 96.7 | 1.4 | 82.6 | 8.1 |
| Claimant Rate of Incapacity Benefit - 2014 | 1.8 | 80.0 | 0.9 | 10.0 | 1.0 |
| Claimant Rate of Income Support - 2014 | 4.4 | 119.6 | 2.3 | 16.8 | 2.0 |
| % of lone parents not in employment - 2011 | 51.9 | 11.2 | 47.5 | 1.6 | 46.7 |
| (ID2010) % of LSOAs in worst 50% nationally - 2010 | -6.4 | 107.7 | 99.2 | 19.5 | 83.0 |
| Average GCSE capped point scores - 2013 | 369.0 | 6.0 | 349.4 | 0.4 | 348.0 |
| Unauthorised Absence in All Schools (%) - 2013 | 1.7 | 53.5 | 0.8 | 26.2 | 1.1 |
| % with no qualifications - 2011 | 20.8 | 19.1 | 18.8 | 7.2 | 17.5 |
| % with Level 4 qualifications and above - 2011 | 44.4 | 25.1 | 39.1 | 10.1 | 35.5 |
| A-Level Average Point Score Per Student - 2012/13 | 715.3 | 5.7 | 668.4 | 1.3 | 676.9 |
| A-Level Average Point Score Per Entry; 2012/13 | 215.0 | 3.1 | 210.8 | 1.1 | 208.5 |
| Violence against the person rate - 2013/14 | 1.2 | 92.5 | 10.5 | 35.6 | 16.3 |
| Robbery rate - 2013/14 | 1.6 | 31.8 | 0.1 | 94.7 | 2.3 |
| Theft and Handling rate - 2013/14 | -3.5 | 113.7 | 11.4 | 55.6 | 25.6 |
| Criminal Damage rate - 2013/14 | 9.1 | 43.8 | 5.9 | 6.6 | 6.3 |
| % area that is open space - 2014 | 30.1 | 28.3 | 19.3 | 17.9 | 23.5 |
| Cars per household - 2011 | 1.6 | 99.4 | 0.5 | 35.0 | 0.8 |
| % travel by bicycle to work - 2011 | 12.0 | 343.9 | 3.0 | 12.5 | 2.7 |
| Turnout at Mayoral election - 2012 | 38.1 | 11.5 | 35.0 | 2.3 | 34.2 |

**Table 4.2:** Median estimation with 22 ciphertexts ($d = 2$, $w = 11$, $\epsilon, \delta = 0.25$) and 165 ciphertexts ($d = 3$, $w = 55$, $\epsilon, \delta = 0.05$) on the London Atlas Dataset.

## Chapter 5

# Privacy-Preserving Data Release with Generative Neural Networks

In Chapter 4, we showed how to train simple machine learning models from aggregate data gathered from many sources.

We now present a novel approach supporting the privacy-preserving release of generative models that is a mixture of $k$ generative neural networks. These networks are trained together and collectively learn the generator distribution of a dataset. The data is first divided into $k$ clusters using a differentially private clustering approach, then each cluster is given to a separate generative neural network, such as Restricted Boltzmann Machines (RBM) [86] or Variational Autoencoders (VAE) [115] (cf. Section 2.3.4). These networks are trained only on their own cluster using differentially private gradient descent.

Training distinct generative models on different partitions of the dataset has several benefits. First, multiple models can generate more accurate synthetic samples than a single model trained on the whole dataset, as each neural network is trained only on similar data samples. This prevents the mixture model to generate unrealistic synthetic samples which may arise from the implausible combination of multiple very different clusters. This scenario is much

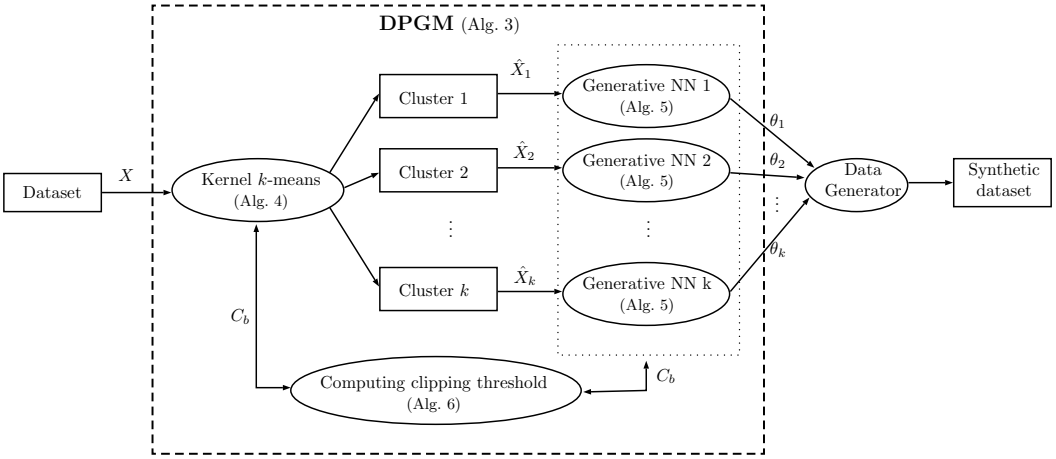



**Figure 5.1:** Overview of our differentially private generative model (DPGM) algorithm.

more likely when the training is perturbed to guarantee differential privacy. Second, each neural network models a different component of the generator distribution, and hence learns any specifics of a cluster faster than a single model. In other words, a single model would need more training epochs than a mixture of generative models to achieve a comparably rich representation of the clusters. As each iteration of the learning algorithm requires some perturbation to guarantee privacy, a mixture model needs less noise which eventually yields more accurate model parameters.

## 5.1 Differentially Private Generative Model (DPGM)

In this section, we present our Differentially Private Generative Model (DPGM) approach, which is detailed in Algorithm 3 and illustrated in Figure 5.1. In the rest of the chapter, we use the following notation: $\mathbb{I}$ denotes a universe of items (e.g., set of visited locations, pixels in an image, etc.), where $|\mathbb{I}| = m$. A dataset $X \subseteq 2^{\mathbb{I}}$ is the ensemble of all items of some set of individuals. A record, which is a non-empty subset of $\mathbb{I}$, refers to all items of an individual from $X$ and is represented by a binary vector $x$ of size $m$. Table 5.1 summarizes notation and symbols used throughout the chapter.

| Symbol | Description |
|---|---|
| $x$ | binary vector |
| $X$ | dataset |
| $k$ | number of $k$-means clusters |
| $T_{\mathcal{K}}$ | $k$-means iterations |
| $T_{\mathcal{G}}$ | SGD iterations |
| $\hat{X}_1, \ldots, \hat{X}_k$ | data clusters |
| $\theta_1, \ldots, \theta_k$ | generative models |
| $\hat{c}_1, \ldots, \hat{c}_k$ | noisy cluster centers |
| $\sigma_{\mathcal{C}}, \sigma_{\mathcal{K}}, \sigma_{\mathcal{G}}$ | noise scales |
| $C_{\max}$ | max. norm bound |
| $w$ | max. number of discretized norm bounds |
| $\kappa$ | kernel function |
| z | randomized Fourier feature map |
| $d$ | number of features |
| $C_b$ | clipping threshold |
| $\mathcal{L}$ | loss function |
| $\eta$ | learning rate |
| $L$ | batch size |

**Table 5.1:** Notation and symbols used in this chapter.

A dataset $X = \{x_1, \ldots, x_N\}$ is first partitioned into $k$ clusters, denoted by $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$, which are in turn used to train $k$ distinct generative models, where the parameters of the resulting models are denoted, respectively, by $\theta_1, \theta_2, \ldots, \theta_k$. Data samples are similar within a cluster, thus, generative models simultaneously trained on each partition converge faster than a single model trained on the whole dataset $X$. As $\theta_1, \theta_2, \ldots, \theta_k$ are learned using perturbed gradient descent, they can be released and used to generate synthetic data using the $k$ generative models.

Our learning approach involves two main steps:

**Algorithm 3:** DPGM: Differentially Private Generative Model

**Input:** *Dataset:* $X = \{x_1, \ldots, x_N\}$, *# of custers:* $k$, *k-means iterations:* $T_{\mathcal{K}}$, *SGD iterations:* $T_{\mathcal{S}}$, *Noise scales:* $\sigma_{\mathcal{C}}, \sigma_{\mathcal{K}}, \sigma_{\mathcal{G}}$

1 **Cluster data records in** $X$**:**

2 $\quad \{\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k\} = \text{DPkmeans}(k, T_{\mathcal{K}}, X, \sigma_{\mathcal{C}}, \sigma_{\mathcal{K}})$ ▷ see Algorithm 4

3 **Initialize** $\theta_1, \theta_2, \ldots, \theta_k$ randomly

4 **for** $t \in [T_{\mathcal{S}}]$ **do**

5 $\quad$ Select $(\hat{X}_b, \theta_b) \in \{(\hat{X}_1, \theta_1), \ldots, (\hat{X}_k, \theta_k)\}$ with probability $|\hat{X}_b|/|X|$

6 $\quad$ **Update parameters of model** $\theta_b$**:**

7 $\quad\quad \theta_b = \text{DP-SGD}(\hat{X}_b, \theta_b, \sigma_{\mathcal{C}}, \sigma_{\mathcal{G}})$ ▷ see Algorithm 5

**Output:** $\theta_1, \theta_2, \ldots, \theta_k$

1. Records in $X$ are clustered in a random feature space using differentially private kernel $k$-means (see Section 5.1.1) into clusters $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$; and

2. A generative model (e.g., RBM [86] or VAE [115]) with parameter $\theta_i$ is trained on cluster $\hat{X}_i$ (see Section 5.1.2) using differentially private gradient descent, where the training data are composed of the records of $\hat{X}_i$.

In each SGD iteration (Line 5-7 in Algorithm 3), a model $\theta_b$ is chosen uniformly at random along with corresponding training data $\hat{X}_b$, and a single SGD iteration is performed to update $\theta_b$ using a random sample $B$ of $\hat{X}_b$ with size $L$ (Line 7 in Algorithm 3). The output of our algorithm are the parameters of the trained generative models, i.e., $\theta_1, \theta_2, \ldots, \theta_k$. Finally, these privately trained models can be used to generate synthetic records which resemble the original ones, i.e., preserve their general characteristics that are not specific to any single individual (as per $\varepsilon$ and $\delta$ discussed in Section 5.2).

### 5.1.1 Private kernel k-means

In Section 2.3.3, we outlined the kernel $k$-means algorithm. We now discuss our private kernel $k$-means algorithm, presented in Algorithm 4.

It first transforms the data $X$ into a low-dimensional representation $X' = \{z(x_1), \ldots, z(x_N)\}$ using randomized Fourier feature map $z : \mathbb{R}^m \to \mathbb{R}^d$ [177], and then applies standard differentially private $k$-means [29] on these low-dimensional features. We rely on randomized Fourier features as they represent one of the most popular techniques for scaling up kernel methods with impressive empirical results [16].

Specifically, $z : \mathbb{R}^m \to \mathbb{R}^d$ is defined as:

$$z(x) = \sqrt{\frac{2}{d}} \left[\cos(\langle \tilde{w}_1, x\rangle + \tilde{b}_1), \ldots, \cos(\langle \tilde{w}_d, x\rangle + \tilde{b}_d)\right] \tag{5.1}$$

where each $\tilde{w}_i \in \mathbb{R}^m$ is drawn independently from $p(\tilde{w}) = \frac{1}{2\pi} \int_{\mathbb{R}^m} \exp(-j\langle \tilde{w}, x\rangle)\kappa(\tilde{w})dx$, i.e., $p(\tilde{w})$ is the Fourier transform of kernel function $\kappa$, and $\tilde{b}_i \in \mathbb{R}$ is chosen from $[0, 2\pi)$ uniformly at random. In particular, Bochner's theorem implies that $p(\tilde{w})$ is a valid probability density function, if $\kappa$ is continuous, positive-definite, and shift-invariant kernel. Hence:

$$\begin{aligned}\kappa(x, y) = \kappa(x - y) &= \int_{\mathbb{R}^m} \exp(j\langle \tilde{w}, x - y\rangle) p(\tilde{w}) d\tilde{w} \\ &= \mathbb{E}_{\tilde{w},\tilde{b}}[\langle \sqrt{2}\cos(\langle \tilde{w}, x\rangle + \tilde{b}), \sqrt{2}\cos(\langle \tilde{w}, y\rangle + \tilde{b})\rangle]\end{aligned}$$

where the expectation is approximated with the empirical mean over $d$ randomly chosen values of $\tilde{w}$ and $\tilde{b}$ [177].

Standard DP $k$-means [29] releases the noisy cluster centers which are computed iteratively using a noisy variant of Lloyd's algorithm; in each iteration, gaussian noise with scale $\sqrt{2}\sigma_{\mathcal{K}}$ is added to the size of all clusters, and with scale $\sqrt{2}\sigma_{\mathcal{K}} C_b$ to the sum of all cluster members in each cluster. These noisy values are used to compute the noisy cluster centers $\{\hat{c}_1, \ldots, \hat{c}_k\}$.

At the beginning, we initialize clusters centers to random records drawn from *publicly available* non-sensitive data which are generated by the same distribution as the sensitive data. We only need $k$ representative samples, and such public datasets already exist for images, location, and medical data. To determine the scale of the gaussian noise, the $L_2$-sensitivity of the cluster size and that of the sum of norms must be known within each cluster. Although

**Algorithm 4:** DPkmeans: Private kernel $k$-means with Random Fourier Features

**Input:** *Data:* $X = \{x_1, \ldots, x_N\}$, *Cluster number:* $k$, *Iterations:* $T_{\mathcal{K}}$, *Feature number:* $d$, *Kernel function:* $\kappa$, *Noise scales:* $\sigma_{\mathcal{C}}, \sigma_{\mathcal{K}}$

**Compute Features:**

$\tilde{w}_i \sim_{\text{iid}} p(\tilde{w})$ for $i \in [1, d]$, where $p(\tilde{w}) = \frac{1}{2\pi} \int_{\mathbb{R}^m} \exp(-j\langle \tilde{w}, x \rangle) \kappa(\tilde{w}) dx$

$\tilde{b}_i \sim_{\text{iid}} \mathcal{U}[0, 2\pi]$ for $i \in [1, d]$

$X' \leftarrow \{z(x_1), \ldots, z(x_N)\}$, where $z(x) = \sqrt{2/d}[\cos(\langle \tilde{w}_1, x \rangle + \tilde{b}_1), \ldots, \cos(\langle \tilde{w}_d, x \rangle + \tilde{b}_d)]$

**Clip Features:**

$C_b \leftarrow \text{DPNorm}(X', \sigma_{\mathcal{C}})$ ▷ see Algorithm 6

$\hat{X}' \leftarrow \{\hat{z}(x_1), \ldots, \hat{z}(x_N)\}$, where $\hat{z}(x_i) = z(x_i) / \max(1, ||z(x_i)||_2 / C_b)$

**Initialize** cluster centers $\hat{c}_1, \hat{c}_2, \ldots, \hat{c}_k$ on public data

**for** $t \in [1, T_{\mathcal{K}}]$ **do**

  **for** $i \in [1, k]$ **do**

    **Assign:**

      $\hat{X}_i \leftarrow \{x : \arg\min_j ||\hat{z}(x) - \hat{c}_j||_2^2 = i\}$

    **Update:**

      $\hat{n}_i \leftarrow |\hat{X}_i| + \mathcal{N}(0, \sqrt{2}\sigma_{\mathcal{K}})$

      $\hat{c}_i \leftarrow 1/\hat{n}_i \left( \sum_{x \in \hat{X}_i} \hat{z}(x) + \mathcal{N}(0, \sqrt{2} C_b \sigma_{\mathcal{K}} I) \right)$

**Output:** $\hat{X}_1, \hat{X}_2, \ldots, \hat{X}_k$

the $L_2$-sensitivity of the set of cluster size is always $\sqrt{2}$ (a single record can change the size of at most 2 clusters), such *a priori* bound does not exist for the $L_2$-norm of the feature vectors in general. Hence, we need to clip all feature vectors in $L_2$-norm before applying standard DP $k$-means, where the clipping threshold $C_b$ should be set to the average norm of the feature vectors (i.e., $(1/N) \sum_{x \in X} ||z(x_i)||_2$) and is approximated by Algorithm 6. Replacing $z(x_i)$ with $\hat{z}(x_i) = z(x_i) / \max(1, ||z(x_i)||_2 / C_b)$ guarantees that all feature vectors are kept as long as their norm is less then $C_b$, or they are scaled down to have a norm of $C_b$.

Nevertheless, for kernel functions like the Radial Basis Function (RBF)[1], a small norm bound $C_b$ can be used (see Theorem 3). This bound is constant for any input data and feature size independently of the width $\gamma$ of the RBF kernel. Thus, as opposed to standard $k$-means [29], our approach can detect linearly non-separable clusters, and, used with RBF kernel, add constant noise to feature vectors independently of their size $d$.

**Theorem 3.** *If* $\kappa(x,y) = \exp(-\gamma||x-y||_2)$*, then* $\mathbb{E}[||z(x)||_2] \leq 1$ *for any* $x \in \{0,1\}^*$ *and* $\gamma$*, where the expectation is taken on the randomness of* $z$*.*

First, we introduce and prove Lemma 2.

**Lemma 2.** *Let* $\mathcal{N}(0,\sigma)$ *be a zero-centered normal random variable with standard deviation* $\sigma$*. Then:*

1. $\mathbb{E}[\cos(\mathcal{N}(0,\sigma))] = \exp(-\sigma^2/2)$ *and*
   $\mathbb{E}[\sin(\mathcal{N}(0,\sigma))] = 0$,
2. $\mathbb{E}[\cos^2(\mathcal{N}(0,\sigma))] = (1+\exp(-2\sigma^2))/2$ *and*
   $\mathbb{E}[\sin^2(\mathcal{N}(0,\sigma))] = (1-\exp(-2\sigma^2))/2$

*Proof of Lemma 2.* Let $\exp(j\mathcal{N}(0,\sigma))$ denote a complex random variable. It follows from the moment generating function of $\mathcal{N}(0,\sigma)$ that:

$$\mathbb{E}[\exp(j\mathcal{N}(0,\sigma))] = \exp((j\sigma)^2/2) = \exp(-\sigma^2/2)$$

which means that:

$$\begin{aligned}\mathbb{E}[\cos(\mathcal{N}(0,\sigma)) + j\sin(\mathcal{N}(0,\sigma))] &= \mathbb{E}[\exp(j\mathcal{N}(0,\sigma))] \\ &= \exp(-\sigma^2/2)\end{aligned}$$

This implies that $\mathbb{E}[\cos(\mathcal{N}(0,\sigma))] = \exp(-\sigma^2/2)$ and $\mathbb{E}[\sin(\mathcal{N}(0,\sigma))] = 0$ due to the linearity of expectation. Hence:

$$\begin{aligned}\mathbb{E}[\cos^2(\mathcal{N}(0,\sigma))] &= \mathbb{E}[(1+\cos(2\mathcal{N}(0,\sigma)))/2] \\ &= (1+\exp(-2\sigma^2))/2\end{aligned}$$

[1] If the kernel function is RBF, i.e., $\kappa(x,y) = \exp(-\gamma||x-y||_2)$, then $p(\tilde{w})$ has zero centered gaussian distribution with standard deviation $2\gamma I$.

and

$$\mathbb{E}[\sin^2(\mathcal{N}(0,\sigma))] = \mathbb{E}[(1-\cos(2\mathcal{N}(0,\sigma)))/2]$$
$$= (1-\exp(-2\sigma^2))/2$$

where we used that $2\mathcal{N}(0,\sigma) = \mathcal{N}(0,2\sigma)$. □

*Proof of Theorem 3.* If $\kappa(x,y) = \exp(-\gamma||x-y||_2)$, then

$$p(\tilde{w}) = \frac{1}{2\pi}\int_{\mathbb{R}^m} \exp(-j\langle \tilde{w}, x\rangle)\kappa(\tilde{w})dx$$

has zero centered gaussian distribution with standard deviation $2\gamma I$.

$$\mathbb{E}[||z(x)||_2] = \mathbb{E}\left[\left((2/d)\sum_{i=1}^{d}\cos^2(\langle \mathcal{N}(0,2\gamma I), x\rangle + \mathcal{U}[0,2\pi])\right)^{\frac{1}{2}}\right]$$
$$\leq \sqrt{\frac{2}{d}}\left(\sum_{i=1}^{d}\mathbb{E}\left[\cos^2(\langle \mathcal{N}(0,2\gamma I), x\rangle + \mathcal{U}[0,2\pi])\right]\right)^{\frac{1}{2}}$$

(by Jensen's inequality and the linearity of expectation)

$$\leq \sqrt{\frac{2}{d}}\left(\sum_{i=1}^{d}\mathbb{E}\left[\cos^2(\langle \mathcal{N}(0,2\gamma I), x\rangle)/2 + \sin^2(\langle \mathcal{N}(0,2\gamma I), x\rangle)/2\right]\right)^{\frac{1}{2}}$$
$$\leq \sqrt{\frac{1}{d}}\left(\sum_{i=1}^{d}\mathbb{E}\left[\cos^2(\mathcal{N}(0,2\gamma\sqrt{||x||_1})\right] + \mathbb{E}\left[\sin^2(\mathcal{N}(0,2\gamma\sqrt{||x||_1})\right]\right)^{\frac{1}{2}}$$

(by Lemma 2)

$$\leq 1$$

where, in the second inequality, we used that $\cos^2(a+b) = \cos^2(a)\cos^2(b) - 2\cos(a)\sin(a)\cos(b)\sin(b) + \sin^2(a)\sin^2(b)$, $\mathbb{E}[\cos(\mathcal{U}[0,2\pi])] = \mathbb{E}[\sin(\mathcal{U}[0,2\pi])] = 0$, $\mathbb{E}[\cos^2(\mathcal{U}[0,2\pi])] = \mathbb{E}[\sin^2(\mathcal{U}[0,2\pi])] = 0.5$. □

Therefore, DP kernel $k$-means has two main advantages over standard DP $k$-means [29]. First, kernel $k$-means can find linearly non-separable clusters. Second, if it is used with RBF kernel, the added noise is independent of the $L_2$-norm of the data records. As we show in Section 5.3, this can lead to much larger clustering accuracy especially for stringent privacy requirements (i.e., for $\varepsilon < 0.5$) even for large dimensional data.

### 5.1.2 Private Stochastic Gradient Descent

We now present our private SGD technique, summarized in Algorithm 5, considering a single SGD batch iteration. Our starting point is the work by Abadi et al. [1]: similar to theirs, our solution provides differential privacy to the training data by first clipping the norm of the gradient update of each record, and then perturbing these clipped gradients by the Gaussian mechanism. However, we achieve better accuracy as the clipping threshold is selected adaptively in each SGD iteration.

In particular, in each SGD iteration, we also (1) compute the gradient of the loss function $\mathcal{L}$ on a random subset $B$ of records (denoted as "batch") in Line 2 of Algorithm 5, (2) clip the $L_2$ norm of the gradient of each record in $B$ to have a norm at most $C_b$ (in Lines 3-6), (3) add gaussian noise $\mathcal{N}(0, \sqrt{2}\sigma_{\mathcal{G}} C_b I)$ to the average of these clipped gradient updates (Line 7), and finally (4) perform the descent step (Line 8). At the end, the updated model parameters $\theta$ are returned. A complete training epoch on the *whole* dataset $X$ consists of $(|X|/L)$ SGD iterations, which are required to process all records in every cluster on average. Indeed, each record in a cluster $\hat{X}_b$ is selected with probability $(|\hat{X}_b|/\sum_{i=1}^{k}|\hat{X}_i|) \times (L/||\hat{X}_b|) = L/|X|$, where $\sum_{i=1}^{k}|\hat{X}_i| = |X|$. Notice that the $L_2$-sensitivity of $\sum_i \hat{g}(x_i)$ is $\sqrt{2}C_b$, as the norm of every $\hat{g}(x_i)$ is at most $C_b$, and one record can change at most two clusters.

### 5.1.3 Adaptive selection of the norm bound

Both our private kernel $k$-means (in Line 6 of Algorithm 4) and private SGD method (in Line 5 of Algorithm 5) require the differentially private computation of the average $L_2$-norm in a given set of records, which is then used as the clipping threshold $C_b$ in both algorithms. For this purpose, these algorithms invoke DPNorm which is detailed in Algorithm 6.

In fact, our SGD technique differs from the original private SGD method by Abadi et al. [1] in the selection of the norm bound $C_b$ (in Lines 4-6 of Algorithm 5). In the original approach [1], $C_b$ is provided as input to the private SGD and no guideline is given how to compute its value without violating

**Algorithm 5:** Private Stochastic Gradient Descent

**Input:** *Data:* $\hat{X}$, *Model parameters:* weights and biases $\theta$, *Noise scales:* $\sigma_{\mathcal{E}}$, $\sigma_{\mathcal{G}}$, Loss function: $\mathcal{L}(\theta) = \frac{1}{|\hat{X}|}\sum_i \mathcal{L}(\theta, x_{c_i})$, Learning rate: $\eta$, Batch size: $L$

1 **Sampling:** Take a random sample $B = \{x_{c_1}, \ldots, x_{c_L}\}$ of $\hat{X}$ with sampling probability $q = L/|\hat{X}|$

2 **Compute Gradient:** For each $x_{c_i} \in B$, compute $g(x_{c_i}) \leftarrow \nabla\mathcal{L}(\theta, x_{c_i})$

3 **Clip Gradient:**

4 $B' \leftarrow \{g(x_{c_i}), \ldots, g(x_{c_L})\}$

5 $C_b \leftarrow \text{DPNorm}(B', \sigma_{\mathcal{E}})$ ▷ see Algorithm 6

6 $\hat{g}(x_{c_i}) \leftarrow g(x_{c_i}) / \max\left(1, \frac{||g(x_{c_i})||_2}{C_b}\right)$

7 **Add noise:** $\tilde{g} \leftarrow \frac{1}{L}\left(\sum_{i=1}^{L} \hat{g}(x_{c_i}) + \mathcal{N}(0, \sqrt{2}\sigma_{\mathcal{G}} C_b I)\right)$

8 **Descent:** $\theta \leftarrow \theta - \eta\tilde{g}$

**Output:** $\theta$

differential privacy. Moreover, the selection of the norm bound $C_b$ has a large impact on the performance of the private SGD in general. If $C_b$ is too small, there will be slow convergence. Conversely, if it is too large, unnecessarily large gaussian noise will be introduced on the gradient update. Intuitively, $C_b$ should be adjusted so that $||g(x_{c_i})||_2 \approx C_b$ for each record $x_{c_i}$. This guarantees that the contribution of $x_{c_i}$ to $\tilde{g}$ is maximally preserved with the smallest relative error. Hence, instead of fixing $C_b$ for the whole training, we aim to compute $C_b$ adaptively for each batch as $C_b = (1/L)\sum_i ||g(x_{c_i})||_2$. This adaptive approach would ensure fast convergence with small error, and also adapt to the gradient update of every batch. Indeed, SGD is iterative, so the gradient update $\tilde{g}$ of a batch/iteration depends on that of the previous batch/iteration, which means that $(1/L)\sum_i ||g(x_{c_i})||_2$ is different for each batch.

In DPNorm (see Algorithm 6), the computation of the average norm in a set $B$ of records is randomized to guarantee privacy. A naive solution is to add gaussian noise to this average, i.e., $C_b = (1/L)\sum_{x \in B} ||x||_2 + \mathcal{N}(0, s \cdot \sigma'/L)$,

**Algorithm 6:** DPNorm: Private Approximation of Average Norm

**Input:** *Data:* $B = \{x_{c_1}, \ldots, x_{c_L}\}$, *Noise scale:* $\sigma_{\mathcal{C}}$, Max. norm bound: $C_{\max}$, Max. number of discretized norm bounds: $w$

1 $C_j \leftarrow j \cdot C_{\max}/w$ for $0 \leq j \leq w$

2 $C_b \leftarrow \arg\max_{j \geq 1}\{t_j + \mathcal{N}(0, \sqrt{2}\sigma_{\mathcal{C}})\}$, where $t_j = |\{x \in B : C_{j-1} < ||g(x)||_2 \leq C_j\}|$

**Output:** $C_b$

where $s \geq \max_{x \in B} ||x||_2$. However, $\max_{x \in B} ||x||_2$ is data-dependent and can be too large if there are outliers in $B$. Instead, we approximate $C_b$ such that its value is close to the norm of many records in $B$, i.e., it is a good approximator of $(1/L)\sum_{x \in B} ||x||_2$. In particular, we discretize the domain of $C_b$ by dividing $(0, C_{\max})$ uniformly into $w$ intervals (Line 1 of Algorithm 6). Then, we use the Gaussian mechanism (Line 2 of Algorithm 6) to select among the upper bounds $C_j = jC_{\max}/w$ of these intervals $(0 \leq j \leq w)$, which will be the norm bound $C_b$ for $B$. Specifically, we build a histogram where bin $i$ equals the number of records whose gradient norm falls within $(C_{i-1}, C_i]$. Then, the (noisy) mode of this histogram is computed by adding independent gaussian noise $\mathcal{N}(0, \sqrt{2}\sigma_{\mathcal{C}})$ to each count, and selecting the bin which has the greatest noisy count. Note that the $L_2$-sensitivity of the histogram is always bounded by $\sqrt{2}$ no matter how large $\max_{x \in B} ||x||_2$ is.

### 5.1.4 Synthetic data generation

To generate an accurate synthetic dataset, data generation should mimic the training process; in order to generate a synthetic sample, a model with parameter $\theta_i$ is first selected randomly with probability $1/|\hat{X}_i|$, then a synthetic sample is generated using the selected model. This process is repeated until $|X|$ samples are obtained.

The above generation process ensures that low quality models which were not selected in training are also less likely to be used for data generation. In particular, though each model is trained during the same number of epochs

*on its own cluster* in expectation, there is no guarantee that all $k$ models will produce identical quality of synthetic samples due to randomization. Indeed, a cluster can potentially contain dissimilar samples, or it may be too small to be selected in Line 5 of Algorithm 3 and hence fail to converge.

## 5.2 Privacy Analysis

In this section, we present the formal privacy analysis of DPGM. Recall that DPGM is the composition of private kernel $k$-means and private SGD. Let $\mathcal{K}$ denote the private kernel $k$-means algorithm whose output is the noisy mapped cluster centers after $T_{\mathcal{K}}$ clustering iterations (i.e., $\mathcal{K}(X) = \{\hat{c}_1, \ldots, \hat{c}_k\}$). $\mathcal{K}$ is composed of (1) selecting the norm bound using DPNorm and (2) $T_{\mathcal{K}}$ iterations of $k$-means. Let $\mathcal{G}_1$ denote the gaussian mechanism which selects the norm bound as per Section 5.1.3. A single $k$-means iteration is the 2-fold adaptive composition of two gaussian mechanisms $\mathcal{G}_2$ and $\mathcal{G}_3$ (in Lines 14-15 of Algorithm 4), where $\mathcal{G}_2$ perturbs the cluster size (Line 14), while $\mathcal{G}_3$ adds noise to the sum of Fourier features of the cluster members (Line 15). The $L_2$-sensitivity of the size of every clusters is $\sqrt{2}$, as changing a single record can change the size of at most two clusters. Similarly, the $L_2$-sensitivity of the sum of Fourier features of the cluster members is $\sqrt{2}C_b$ as it is detailed in Section 5.1.1.

Since $\mathcal{K}$ is the $T_{\mathcal{K}}$-fold adaptive composition of $T_{\mathcal{K}}$ clustering iterations, it follows from Theorem 2 and Lemma 1 (see Section 2.1.3):

$$\begin{aligned} \beta_{\mathcal{K}}(\lambda) &\leq T_{\mathcal{K}}(\beta_{\mathcal{G}_1}(\lambda) + \beta_{\mathcal{G}_2}(\lambda) + \beta_{\mathcal{G}_3}(\lambda)) \\ &\leq T_{\mathcal{K}}(\lambda^2 + \lambda)(1/4\sigma_{\mathcal{C}}^2 + 1/2\sigma_{\mathcal{K}}^2) \end{aligned} \tag{5.2}$$

Note that if the RBF kernel is used in kernel $k$-means (i.e., $\kappa(x, y) = \exp(-\gamma||x-y||_2)$ in Algorithm 4), then $\beta_{\mathcal{G}_1}(\lambda) = 0$ and $\beta_{\mathcal{K}}(\lambda) \leq T_{\mathcal{K}}(\lambda^2 + \lambda)/2\sigma_{\mathcal{K}}^2$ since $C_b = 1$ is a priori bound on the $L_2$-norm of every feature vector (cf. Theorem 3).

Let $\mathcal{S}_k$ denote the private SGD algorithm whose output is the noisy model parameters after $T_{\mathcal{S}}$ SGD iterations (i.e., $\mathcal{S}(X) = \{\theta_1, \ldots, \theta_k\}$, computed in the

last iteration of Algorithm 3), and the input is the cluster centers $\{\hat{c}_1, \ldots, \hat{c}_k\}$ provided by $\mathcal{K}$. At the very beginning, $\mathcal{S}$ assigns each record to its closest cluster center in feature space to obtain $k$ non-overlapping training sets (this is implemented by the last iteration in Algorithm 4). Changing a single record alters at most a single record in at most 2 training sets (clusters), as the modified record can be moved from one to another training set. Since all training sets are non-overlapping, each record is selected in an SGD iteration with probability $q = (|\hat{X}_b|/|X|) \times (L/|\hat{X}_b|) = L/|X|$ for any $k$. Moreover, each of the $k$ models are trained independently, so $\beta_{\mathcal{S}_k}(\lambda) \leq \beta_{\mathcal{S}_1}(\lambda)$, where $\mathcal{S}_1$ denotes the case when $k = 1$ (i.e., a single model is trained on the whole dataset $X$ during $T_{\mathcal{S}}$ epochs).

The complete SGD training of $\mathcal{S}_1$ (Lines 4-7 in Algorithm 3) is the $T_{\mathcal{S}}$-fold adaptive composition of $T_{\mathcal{S}}$ SGD iterations, where we jointly use two perturbation mechanisms $\mathcal{G}_4$ and $\mathcal{G}_5$ in each iteration; $\mathcal{G}_4$ selects a batch uniformly at random and computes the norm bound $C_b$ (Line 5 of Algorithm 5) for this batch, then $\mathcal{G}_5$ selects the same batch and perturbs its gradient updates with gaussian noise whose magnitude is calibrated to $C_b$ (Line 7 of Algorithm 5).

The composition of these two mechanisms uses independent source of randomness through different SGD iterations, hence we can use Theorem 2 to quantify the overall privacy. However, within a single iteration, $\mathcal{G}_4$ and $\mathcal{G}_5$ do not use independent source of randomness, although both mechanisms use independent gaussian noise but select the same batch $B$ from the dataset. The following theorem computes $\beta_{\mathcal{S}_1}(\lambda)$, and is a generalization of Theorem 2 when the component mechanisms can use dependent source of randomness.

**Theorem 4** (General Moments Accountant)**.** *Let* $\beta_{\mathcal{A}_i}(\lambda)$ *be* $\max_{X,X'} \log \mathbb{E}_{O\sim\mathcal{A}(X)}[\exp(\lambda \mathcal{P}(\mathcal{A}, X, X', O))]$*, and* $\mathcal{A}_{1:k}$ *be the* $k$*-fold adaptive composition of* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$*. Then:*

1. $\beta_{\mathcal{A}_{1:k}}(\lambda) \leq \sum_{i=1}^{k} j_i \beta_{\mathcal{A}_i}(\lambda/j_i)$
2. $\mathcal{A}_{1:k}$ *is* $(\varepsilon, \min_\lambda \exp(\sum_{i=1}^{k} j_i \cdot \beta_{\mathcal{A}_i}(\lambda/j_i) - \lambda\varepsilon))$*-DP*

*for any* $\sum_{i=1}^{k} j_i = 1$*, where* $j_i > 0$ *and* $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ *can use dependent coin tosses.*

*Proof.* We extend the proof of Theorem 2 in [1] to the case when the composite mechanism $\mathcal{A}_{1:k}$ consists of dependent mechanisms.

Let $\mathcal{A}_{1:k}$ denote the composition of $\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_k$ and $O = (O_1, O_2, \ldots, O_k)$. Recall that, from Definition 1, $\mathcal{A}_{1:k}$ is $(\varepsilon, \delta)$-DP, if $\Pr_{O \sim \mathcal{A}_{1:k}(X)}[\mathcal{P}(\mathcal{A}_{1:k}, X, X', O) > \varepsilon] \leq \delta$. Then:

$$
\begin{aligned}
&\mathcal{P}(\mathcal{A}_{1:k}, X, X', O) \\
&\quad = \log \frac{\Pr[\mathcal{A}_{1:k}(X) = O]}{\Pr[\mathcal{A}_{1:k}(X') = O]} \\
&\quad = \log \prod_{i=1}^{k} \frac{\Pr[\mathcal{A}_i(X) = O_i | \mathcal{A}_{i-1}(X) = O_{i-1}, \ldots, \mathcal{A}_1(X) = O_1]}{\Pr[\mathcal{A}_i(X') = O_i | \mathcal{A}_{i-1}(X') = O_{i-1}, \ldots, \mathcal{A}_1(X') = O_1]} \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{(by the Chain rule)} \\
&\quad = \sum_{i=1}^{k} \log \frac{\Pr[\mathcal{A}_i(X) = O_i | \mathcal{A}_{i-1}(X) = O_{i-1}, \ldots, \mathcal{A}_1(X) = O_1]}{\Pr[\mathcal{A}_i(X') = O_i | \mathcal{A}_{i-1}(X') = O_{i-1}, \ldots, \mathcal{A}_1(X') = O_1]} \\
&\quad = \sum_{i=1}^{k} \mathcal{P}(\mathcal{A}_i, X, X', O_i) \qquad (5.3)
\end{aligned}
$$

for any neighboring datasets $X$ and $X'$. Hence,

$$
\begin{aligned}
\beta_{\mathcal{A}_{1:k}}(\lambda) &= \max_{X, X'} \log \mathbb{E}_{O \sim \mathcal{A}(X)}[\exp(\lambda \mathcal{P}(\mathcal{A}_{1:k}, X, X', O))] \\
&= \max_{X, X'} \log \mathbb{E}_{O \sim \mathcal{A}(X)} \left[ \exp\left( \lambda \sum_{i=1}^{k} \mathcal{P}(\mathcal{A}_i, X, X', O_i) \right) \right] \quad \text{(by Eq. (5.3))} \\
&= \max_{X, X'} \log \mathbb{E}_{O \sim \mathcal{A}(X)} \left[ \prod_{i=1}^{k} \exp\left( \lambda \mathcal{P}(\mathcal{A}_i, X, X', O_i) \right) \right] \\
&\leq \max_{X, X'} \log \prod_{i=1}^{k} \left( \mathbb{E}_{O_i \sim \mathcal{A}_i(X)} \left[ \exp\left( \lambda \mathcal{P}(\mathcal{A}_i, X, X', O_i)/j_i \right) \right] \right)^{j_i} \\
&\qquad\qquad\qquad\qquad \text{(by the generalization of Hölder's inequality)} \\
&\leq \max_{X, X'} \sum_{i=1}^{k} j_i \log \left( \mathbb{E}_{O_i \sim \mathcal{A}_i(X)} \left[ \exp\left( \lambda \mathcal{P}(\mathcal{A}_i, X, X', O_i)/j_i \right) \right] \right) \\
&\leq \sum_{i=1}^{k} j_i \max_{X, X'} \log \left( \mathbb{E}_{O_i \sim \mathcal{A}_i(X)} \left[ \exp\left( \lambda \mathcal{P}(\mathcal{A}_i, X, X', O_i)/j_i \right) \right] \right) \\
&\leq \sum_{i=1}^{k} j_i \beta_{\mathcal{A}_i}(\lambda / j_i) \qquad (5.4)
\end{aligned}
$$

where we can apply the generalization of Hölder's inequality in the first inequality due to the fact that $\exp(\cdot)$ is always positive. Therefore,

$$\begin{aligned}
&\Pr[\mathcal{P}(\mathcal{A}_{1:k}, X, X', O) \geq \varepsilon] = \Pr[\exp(\lambda \mathcal{P}(\mathcal{A}_{1:k}, X, X', O)) \geq \exp(\lambda\varepsilon)] \\
&\leq \mathbb{E}_{O\sim\mathcal{A}(X)}[\exp(\lambda \mathcal{P}(\mathcal{A}_{1:k}, X, X', O))] / \exp(\lambda\varepsilon) \qquad \text{(by Markov's inequality)} \\
&\leq \exp(\beta_{\mathcal{A}_{1:k}}(\lambda) - \lambda\varepsilon) \leq \exp\left(\sum_{i=1}^{k} j_i \beta_{\mathcal{A}_i}(\lambda/j_i) - \lambda\varepsilon\right) \qquad \text{(by Eq. (5.4))}
\end{aligned}$$

The claim follows from Definition 1. □

Therefore, it follows from Theorem 2 and 4 that:

$$\begin{aligned}
\beta_{\mathcal{S}_k}(\lambda) &\leq \beta_{\mathcal{S}_1}(\lambda) \\
&\leq T_{\mathcal{S}} \cdot \min_{j_1, j_2 \in (0,1): j_1 + j_2 = 1} (j_1 \beta_{\mathcal{G}_4}(\lambda/j_1) + j_2 \beta_{\mathcal{G}_5}(\lambda/j_2)) \qquad (5.5)
\end{aligned}$$

We compute $\beta_{\mathcal{G}_4}(\lambda)$ and $\beta_{\mathcal{G}_5}(\lambda)$ similarly to [1]. That is, let $\mu_0(x|\sigma) = g(x|\sigma)$ and $\mu_1(x|\sigma) = (1-q)g(x|\sigma) + qg(x-1|\sigma)$, where $q = L/|X|$ is the probability that a record is included in the batch $B$ of an SGD iteration and $g(x|\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-x^2/2\sigma^2}$. Then, it holds:

$$\begin{aligned}
\beta_{\mathcal{G}_3}(\lambda) &= \log \max(E_1(\lambda, \sigma_{\mathcal{C}}), E_2(\lambda, \sigma_{\mathcal{C}})) \\
\beta_{\mathcal{G}_4}(\lambda) &= \log \max(E_1(\lambda, \sigma_{\mathcal{G}}), E_2(\lambda, \sigma_{\mathcal{G}}))
\end{aligned}$$

where

$$\begin{aligned}
E_1(\lambda, \sigma) &= \int_{-\infty}^{\infty} \mu_0(x|\sigma) \cdot \left(\frac{\mu_0(x|\sigma)}{\mu_1(x|\sigma)}\right)^{\lambda} dx \\
E_2(\lambda, \sigma) &= \int_{-\infty}^{\infty} \mu_1(x|\sigma) \cdot \left(\frac{\mu_1(x|\sigma)}{\mu_0(x|\sigma)}\right)^{\lambda} dx
\end{aligned}$$

The next theorem immediately follows from Theorem 2 and Theorem 4.

**Theorem 5.** *Our differentially private generative model (DPGM) is*

$$(\min_{\lambda} \left(\beta_{\mathcal{K}}(\lambda) + \beta_{\mathcal{S}_k}(\lambda) - \log\delta\right)/\lambda, \delta)$$

*-differentially private for any fixed $\delta$, where $\beta_{\mathcal{K}}(\lambda)$ and $\beta_{\mathcal{S}_k}(\lambda)$ are defined in Eq. 5.2 and 5.5.*

| ***Dataset*** | $\lvert X\rvert$ | $\lvert\mathbb{I}\rvert = m$ | $\max \lvert\lvert x\rvert\rvert_1$ | avg $\lvert\lvert x\rvert\rvert_1$ |
|---|---|---|---|---|
| **MNIST** | 60,000 | 784 | 311.69 | 102.44 |
| **CDR** | 4,427,486 | 1303 | 422 | 11.42 |
| **TRANSIT** | 1,200,000 | 342 | 57 | 5.26 |

**Table 5.2:** The datasets used in our experiments: MNIST (images), CDR (call detail records), and TRANSIT (transport records).

We use the convention that $\delta = 1/|X|$, and compute $\varepsilon$ numerically. Specifically, $\varepsilon = \min_\lambda \left(\beta_{\mathcal{K}}(\lambda) + \beta_{\mathcal{S}_k}(\lambda) - \log\delta\right)/\lambda$ is minimized over integer values of $\lambda$, where $\lambda$ is usually no more than 100 in practice. The computation of $\beta_{\mathcal{G}_3}$ and $\beta_{\mathcal{G}_4}$ are performed through numerical integration, and it suffices to consider 10 different values of $j_1$ and $j_2$ in order to have a sufficiently small value of $j_1\beta_{\mathcal{G}_3}(\lambda/\ell_1) + j_2\beta_{\mathcal{G}_4}(\lambda/\ell_2)$ in Eq. 5.5. Therefore, in practice, given $\delta$, an accurate approximation of $\varepsilon$ can be obtained with negligible overhead.

## 5.3 Experimental Evaluation

In this section, we report the results of an experimental evaluation geared to compute the exact privacy guarantees of DPGM (presented in Algorithm 3). We also analyze its performance in terms of the quality of generated samples as well as counting (linear) queries computed on the synthetic data. Counting queries provide the basis of many data analysis and learning algorithms (see [29] for examples). Finally, we measure the accuracy of our private kernel $k$-means described in Algorithm 4.

### 5.3.1 Experimental setup

**Datasets.** We use three datasets for our evaluations, summarized in Table 5.2. MNIST is a public image dataset [124], which includes $28 \times 28$-pixel images of hand-written digits, a total of $60,000$ samples. We vectorize and binarize each image to have binary data records with size $m = 784$. Throughout our experiments, we assume that each of the $60,000$ records originates from a

different person. We also use an anonymized CDR (Call Detail Record) dataset provided to us by a cell phone operator. For this dataset, $\mathbb{I}$ represents the set of cell towers of the operator in a large city with $|X| = 4,427,486$ customers. We use a simplified version of the dataset, which contains the set of visited cell towers per customer within the administrative region of the city over 128.1 $km^2$, where the total number of towers is $m = 1,303$.

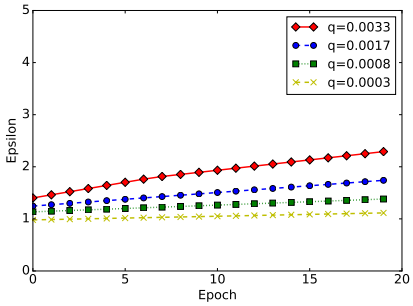
Finally, we experiment with a transit dataset, which we denote as TRANSIT in the rest of the chapter. Due to non-disclosure agreement, we are unable to provide specific details about the dataset, however, we can report that the TRANSIT dataset include the transit history of passengers in the network (with $|X| = 1,200,000$); here, $\mathbb{I}$ represents the set of $m = 342$ stations in a public transportation network.

**Experimental setup.** For RBM, we set the number of hidden units to 200 and the learning rate is 0.01. The biases $b$ and $c$ are initialized to zeros, while the initial values of the weights $W$ are randomly chosen from a zero-mean Gaussian with a standard deviation of 0.01. For VAE, the number of hidden units is set to 200 with single layer encoder and decoder, and a bi-dimensional latent space. We also used the rectifier activation function (ReLu) for all neurons and the Adam optimizer [116]. For our purposes, it is enough to compute $\beta(\lambda)$ for $\lambda \leq 32$. We set the number of the private k-means iterations to 20 and $\delta = 1/|X|$. We also set $C_{\max} = 10$, $w = 100$ (in Algorithm 6), as different values of these parameters do not have a strong impact on the results.

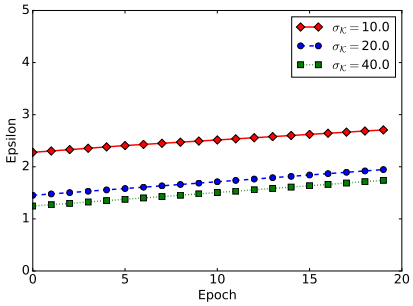
We implement DPGM with both RBM (in C++) and VAE (in Python). Experiments are performed on a workstation running Ubuntu Server 16.04 LTS, with a 3.4 GHz CPU i7-6800K, 32GB RAM, and NVIDIA Titan X GPU card.

### 5.3.2 Results with image dataset

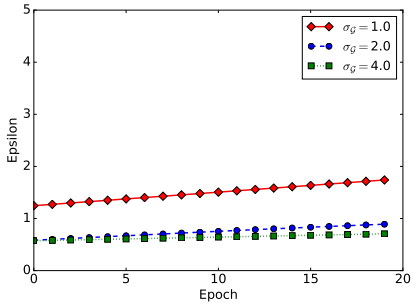
**Privacy guarantees.** We report the privacy loss $\varepsilon$ of DPGM (Algorithm 3) in Figure 5.2 for the MNIST dataset. Recall that $\varepsilon$ is computed from the noise




**(a)** Sampling probability $q$ ($\sigma_{\mathcal{G}} = 1.0$, $\sigma_{\mathcal{K}} = 40.0$)




**(b)** Clustering noise $\sigma_{\mathcal{K}}$ ($\sigma_{\mathcal{G}} = 1.0$, $q = 0.0017$)




**(c)** SGD noise $\sigma_{\mathcal{G}}$ ($\sigma_{\mathcal{K}} = 40.0$, $q = 0.0017$)

**Figure 5.2:** $\varepsilon$ value as a function of the number of SGD training epochs for MNIST ($\delta = 10^{-5}, T_{\mathcal{K}} = 20$)

level $\sigma_{\mathcal{C}}$, $\sigma_{\mathcal{K}}$, and $\sigma_{\mathcal{G}}$, the sampling probability $q$, the number of $k$-means iterations $T_{\mathcal{K}}$, and the number of SGD iterations $T_{\mathcal{S}}$ using Theorem 5. Figure 5.2 shows $\varepsilon$ depending on the number of SGD training epochs, where one epoch consists of $\lceil 1/q \rceil$ SGD iterations. In Figure 5.2(a)–5.2(c), we fix $\sigma_{\mathcal{C}} = 4.0$, and report the value of $\varepsilon$ as a function of the number of epochs. We note that larger sampling probabilities ($q$) and more epochs yield larger values of $\varepsilon$, i.e., worse privacy guarantee. Figure 5.2(b)– 5.2(c) show that larger values of $\sigma_{\mathcal{K}}$ and $\sigma_{\mathcal{G}}$ yield stronger privacy guarantees.

**Clustering accuracy.** Next, in Figure 5.3, we compare the private kernel $k$-means (Algorithm 4) with RBF kernel with standard DP $k$-means [29]. We evaluate the unsupervised clustering accuracy (ACC) [214], where ACC =

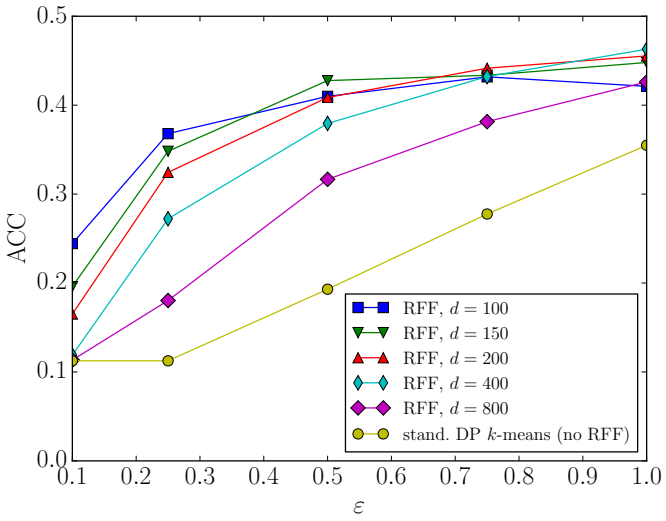
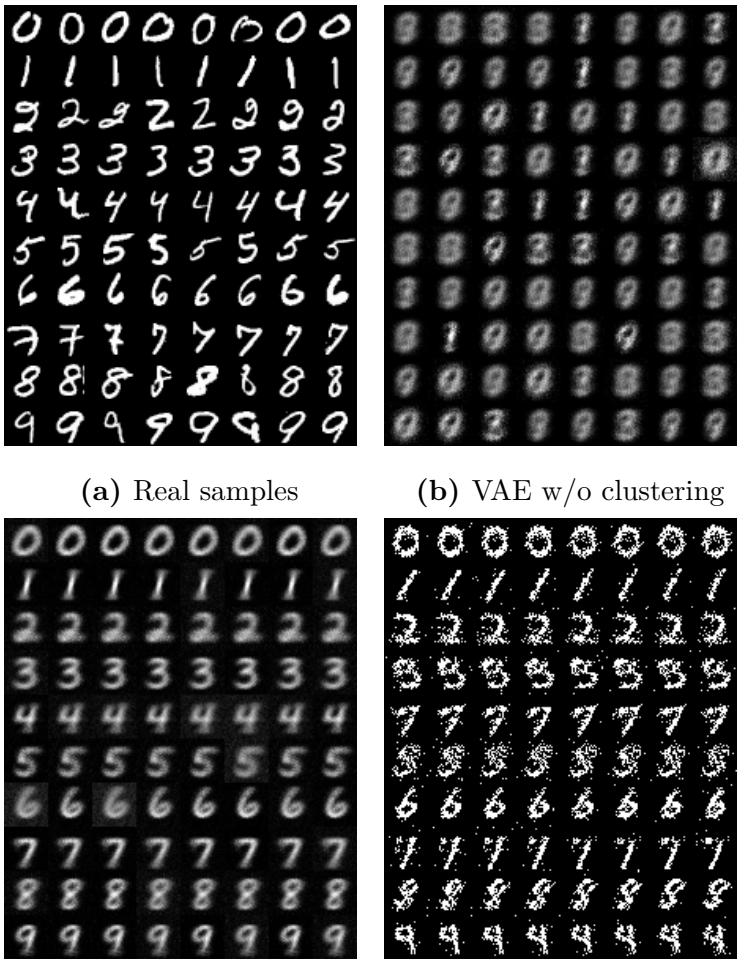



**Figure 5.3:** Clustering accuracy as a function of $\varepsilon$ on MNIST ($\delta = 10^{-5}, T_{\mathcal{K}} = 20$).

$\max_u \frac{|\{x : x \in X \wedge label(x) = u(\mathcal{K}(x))\}|}{|X|}$, $label(x)$ is the ground-truth label of sample $x$[2], $\mathcal{K}(x)$ is the cluster assignment obtained by clustering algorithm $\mathcal{K}$, and $u$ is a one-to-one mapping between cluster assignments and labels. The best mapping can be obtained using the Hungarian algorithm. To make a fair comparison, we fix $C_b$ to $\sqrt{m} = 28$ for standard private $k$-means without RFF features, and $C_b = 1$ for private kernel $k$-means with RFF features based on Theorem 3 – i.e., we do not call DPNorm in either of the algorithms. We compute the clustering accuracy for different values of $d$ depending on $\sigma_{\mathcal{K}}$, which directly yields the privacy bound $\varepsilon$ using Eq. 5.2 and Theorem 2. Finally, we plot the average accuracy over 100 runs as function of $\varepsilon$ in Figure 5.3.[3] Private kernel $k$-means is clearly superior to standard DP $k$-means, as the difference in clustering accuracy can be as large as 20%, especially for smaller values of $\varepsilon$. Shorter RFF features (i.e., smaller $d$) result in larger accuracy for smaller values of $\varepsilon$, whereas the reverse holds for larger $\varepsilon$. The reason is that the clustering error is determined by the trade-off between (1) the perturbation error due to the Gaussian noise, which is added to the cluster centers in Line 15 of Algorithm 4, and (2) the approximation error caused by the low-dimensional

[2]For MNIST, these are digits ranging from 0 to 9.

[3]Standard deviation of accuracy is $< 0.05$ for all values of $\varepsilon$ and $d$.

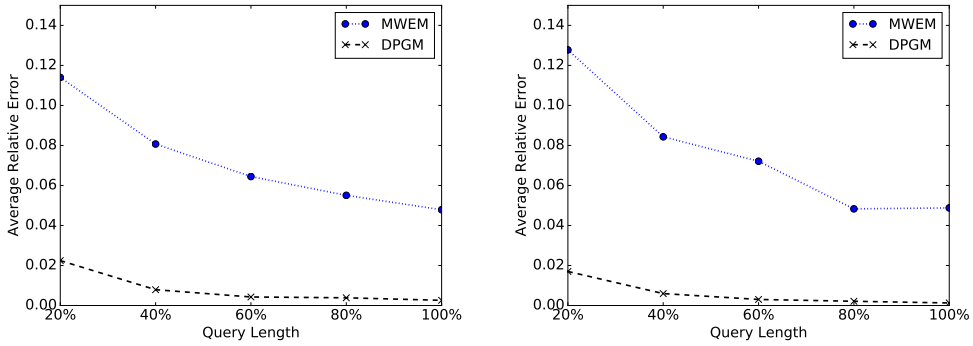


**(a)** Real samples **(b)** VAE w/o clustering

**(c)** VAE with clustering **(d)** RBM with clustering

**Figure 5.4:** Real MNIST samples and samples generated from DPGM with RBM and VAE after 20 epochs ($\varepsilon = 1.74, T_{\mathcal{K}} = 20$). In **(c)** and **(d)**, each row contains 8 samples generated from a cluster.

embedding $z$ in Line 4 of Algorithm 4. In particular, the perturbation error increases if $\varepsilon$ decreases or $d$ increases. Indeed, when the distance $||\hat{z}(x) - \hat{c}_j||_2^2$ to each cluster center $\hat{c}_j$ is computed in Line 12 of Algorithm 4, the total perturbation of this distance value is obtained by aggregating the noise values on each coordinate of $\hat{c}_j$, and hence the perturbation error is proportional to the size $d$ of vector $\hat{c}_j$ as well as to $\varepsilon^{-1}$. On the other hand, larger $d$ decreases the approximation error introduced by $z$. One can find a good trade-off between the approximation and the perturbation error by adjusting $d$ and $\varepsilon$ through



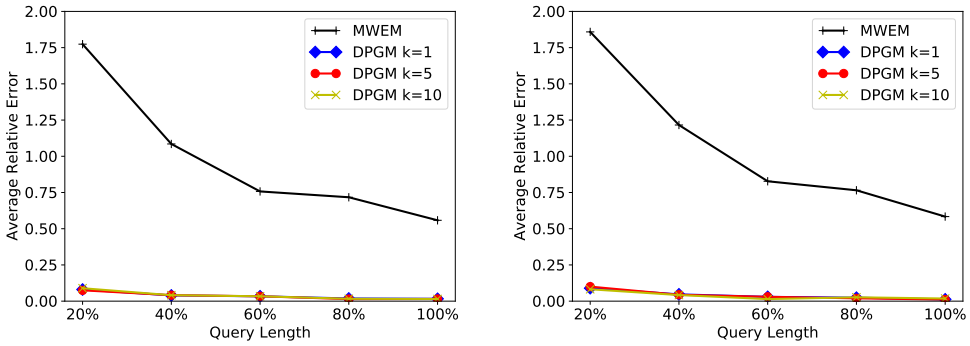

**(a)** $\varepsilon = 2.0$ **(b)** $\varepsilon = 1.0$

**Figure 5.5:** Average relative error vs. $\varepsilon$ for the CDR dataset ($q = 2.2 \cdot 10^{-5}, \delta = 4.4 \cdot 10^{-6}$)

experiments using publicly available data. For the rest of experiments, we set $d$ to 200.

Selecting the optimal number of clusters $k$ for kernel $k$-means can be qualitatively and visually done by relying on dimensionality reduction algorithms (e.g., t-SNE [136]). To this end, one can use public data sampled from the same underlying distribution, and therefore not requiring to make the parameter selection step differentially private. For MNIST we set $k = 10$, while we select only one cluster for the CDR dataset. We investigate the effects of different values of $k$ for the transit dataset.

**Synthetic samples.** As training progresses, the synthetic samples produced by the generative models should resemble the true samples. To evaluate model quality, we show the synthetic samples obtained at epoch 20 in Figure 5.4 from a Restricted Boltzmann Machine and a Variational Autoencoder with $k = 10$ clusters on MNIST. For this experiment, we set $q = 0.0017$ for a final privacy budget $\varepsilon$ of 1.74, and performed $T_{\mathcal{K}} = 20$ clustering iterations before training the generative neural networks. Overall, the samples generated from VAE (Figure 5.4(c)) provide better visual quality than the ones generated from the RBM (Figure 5.4(d)). Note that the samples generated from the VAE without our private clustering technique (Figure 5.4(b)) have bad visual quality.




**(a)** $\varepsilon = 2.0$ **(b)** $\varepsilon = 1.0$

**Figure 5.6:** Average relative error vs. $\varepsilon$ for the transit dataset ($q = 10^{-4}, \delta = 10^{-6}$)

### 5.3.3 Results with CDR and transit dataset

We consider counting queries, which are specified by a predicate function $p : X \to \{0,1\}$ and return the number of users in the dataset which satisfy the given predicate $p$, i.e., $Q_p(X) = \sum_{x \in X} p(x)$. We evaluate the accuracy of counting queries on a synthetic dataset generated by DPGM from our call-data-record (CDR) dataset with roughly 4 million users and the transit dataset with roughly 1 million users (see Table 5.2). A single query is defined by a subset of tower cells, and returns the number of users in $X$ who visited these cells. We compare DPGM with MWEM [93], which is a *de facto* standard differentially private mechanism to answer counting queries.

As done in previous work [213], we measure the utility of a counting query $Q_p$ over the sanitized dataset $\hat{X}$ by its relative error with respect to the actual result over the raw dataset $X$. The relative error of $Q_p$ is thus computed as $\frac{|Q_p(\hat{X}) - Q_p(X)|}{\max\{Q_p(X), s\}}$, where $s$ is a sanity bound that weight the influence of the queries with small selectivities. Following the convention, the sanity bound is set to 0.1% of the dataset size.

First, we examine the relative error of counting queries with respect to privacy loss $\varepsilon$. 1,000 counting queries are randomly generated with different number of tower cells, which we refer as the length of the query. Each query set is divided into 5 subsets such that the query length of the $i$-th subset is

uniformly distributed in $\left[1, \frac{i \cdot \max ||x||_1}{5}\right]$ and each item is randomly drawn from universe of items. Figure 5.5 reports the average relative error for each query set. This shows that our approach clearly outperforms MWEM. We think this is due to the use of advanced generative machine learning models, which learn a better approximation of the real dataset than the Multiplicative Weight update rule of MWEM. Furthermore, our approach relies on the moments accountant [1], which provides a tighter bound on the privacy loss than the one provided by the Exponential Mechanism in MWEM.

The error of DPGM ranges from 0.017 for 20% query length to 0.0012 for 100% when $\varepsilon = 1.0$. Weaker privacy guarantee (larger values of $\varepsilon$) lead to slightly smaller errors (Figure 5.5(b)). By contrast, the error of MWEM ranges from 0.11 to 0.05 even for $\varepsilon = 2$. After clipping each record to have $L_1$-norm $\text{avg}||x||_1 = 12$, the sensitivity of queries is set to 12, and the iterations of the MWEM algorithm is set to 50 [93]. Also note that the synthetic data produced by DPGM allows the evaluation of arbitrary number of type of queries, not only linear counting queries.

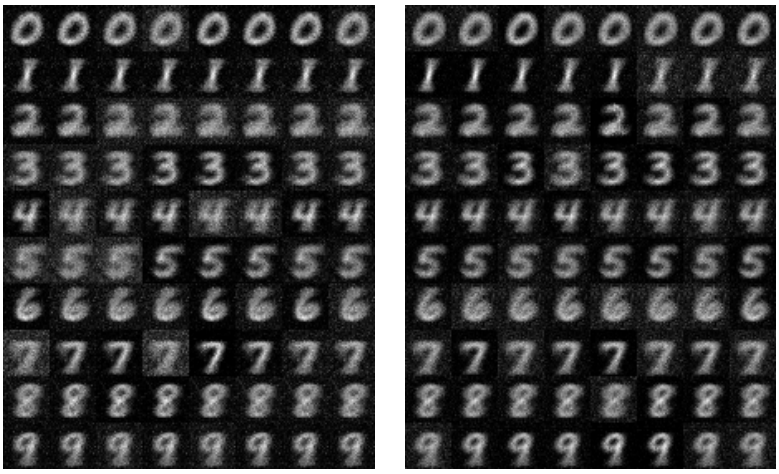
Finally, Figure 5.6 reports the average relative error for the transit dataset with different number of clusters $k$. Our approach, whose average relative error ranges from 0.09 to 0.02, significantly outperforms MWEM. However, the number of clusters does not affect the error of counting queries on transit dataset due to the fact that the private clustering fails to find meaningful clusters for this type of data.

### 5.3.4 Multi-layer Variational Autoencoder

Finally, we report additional results for a VAE with a double layer encoder and decoder. In Figure 5.7, we show the synthetic samples obtained at epoch 20 from a VAE with $k = 10$ clusters on MNIST.

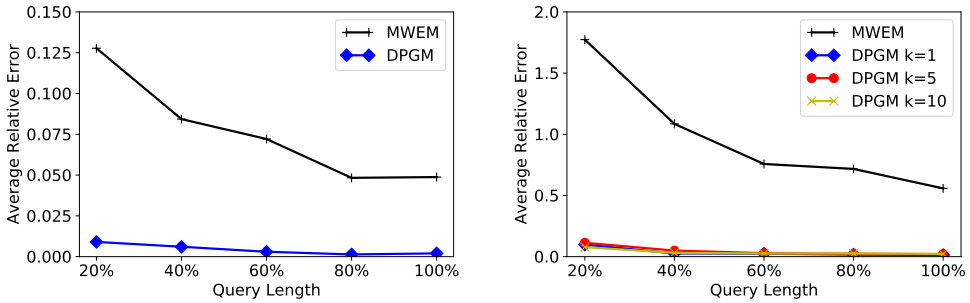
Then, Figure 5.8(a) reports the average relative error for the CDR dataset, while Figure 5.8(b) shows the average relative error for the transit dataset with different number of clusters $k$.

Overall, we can observe that increasing the number of layers, and thus



**(a)** $\varepsilon = 1.74$ **(b)** $\varepsilon = 2.0$

**Figure 5.7:** Samples generated from a double layer VAE after 20 epochs. Each row contains 8 samples generated from a cluster.




**(a)** CDR dataset **(b)** Transit dataset

**Figure 5.8:** Average relative error with $\varepsilon = 1.0$ for the CDR and transit datasets.

the capacity of the VAE, does not lead to better performance for this dataset.

## Chapter 6

# Evaluating Privacy Leakage of Generative Models

As discussed in Section 3.2.3, previous work proposed passive and active membership inference attacks – i.e., the presence of exact data points in training data – against machine learning classifiers [189, 222]. By contrast, in this chapter, we set out to study how generating synthetic samples through generative models may lead to information leakage, hence, to violating privacy of individuals contributing their (sensitive) data to train these models.

More specifically, aiming to perform membership inference, we train a Generative Adversarial Network (GAN), described in Section 2.3.4, on samples generated from the model under attack. Our intuition is that, if a generative model overfits, then a GAN, which combines a discriminator and a generator, should be able to detect this overfitting, even if it is not observable to a human, since the discriminator is trained to learn statistical differences in distributions. We leverage the ability of GANs to classify real and synthetic records in order to recognize differences in the samples generated from the model, on inputs on which it was trained versus those on which it was not.

We test our attacks on several state-of-the-art models such as Deep

The research presented in this chapter was joint work with a fellow PhD student Jamie Hayes. The author equally contributed to the design of the attacks presented in Section 6.1, and lead the work on the experimental evaluation of the attacks on the LFW dataset in Sections 6.2 and 6.3, as well as the evaluation of potential defenses described in Section 6.3.3.

Convolutional GAN (DCGAN) [176], Boundary Equilibrium GAN (BEGAN) [25], and the combination of DCGAN with a Variational Autoencoder (DCGAN+VAE) [122], using datasets with complex representations of faces (LFW), objects (CIFAR-10), and medical images (Diabetic Retinopathy), containing rich details both in the foreground and background. This represents a much more challenging task for the attacker compared to simple datasets such as MNIST, where samples from each class have very similar features.

## 6.1 Attacks Outline

In this section, we present our membership inference attacks against generative models.

### 6.1.1 Threat model

We consider an adversary that aims to infer whether *a single known record* was included in the training set of a generative model. We distinguish between two settings, namely, *black-box* and *white-box* attacks. In the former, the attacker can only make queries to the target model under attack – which we denote as the *target model* – and has no access to the internal parameters of the model; in the latter, they also have access to the parameters of a trained target model. In both settings, we allow the adversary to know the size of the training set, but not its original data-points. Variants of the attack allow the adversary to access some further side information, as discussed below. The accuracy of the attack is measured as the fraction of the records correctly inferred as members of the training set.

Note that, in order to evaluate the accuracy of our attacks, we will consider an attacker attempting to distinguish data-points used to train the target model, thus, we consider an attacker that has access to a dataset they suspect contains the original training records. However, the construction of the attack does *not* depend on access to *any* dataset. We assume the attacker knows the size of the training set – as part of the information included with the target model metadata or leaked following a data breach – but does *not* know how

data-points are split into training and test sets.

In the black-box attack setting, we assume the attacker does not have prior or side information about training records or the target model. In particular, the attack proceeds with **no knowledge** of the following:

1. *Target model parameters and hyper-parameters:* No access to network weights from the trained target model, nor to hyper-parameters such as regularization parameters or number of epochs used to train the target model.

2. *Target model architecture:* The attacker has no knowledge of the architecture of the target model.

3. *Dataset used to train the target model:* No knowledge of data-points used to train the target model, or the *type* of data-points used in training, since this is inferred from sampling the target model at inference time. Note that, by contrast, the membership inference attack on discriminative models by Shokri et al. [189] *does* require some information about the dataset, e.g., the syntactic format of data records used in training, in order to generate synthetic samples used in the attack.

4. *Prediction values:* Shokri et al. [189] show that predictions scores leak information used to perform membership inference attacks. However, due to the very nature of generative models, the attacker cannot generate prediction scores directly from the target model.

### 6.1.2 White-box attack

We now present our white-box attack, illustrated in Figure 6.1. To evaluate the attack, here we assume that an attacker $\mathcal{A}_{wb}$ has access to the trained target model, namely, a GAN – i.e., a generator $G_{target}$ and a discriminator $D_{target}$. We also assume the attacker has access to a dataset, $X = \{x_1, \ldots, x_{m+n}\}$, which they suspect contains data-points used to train

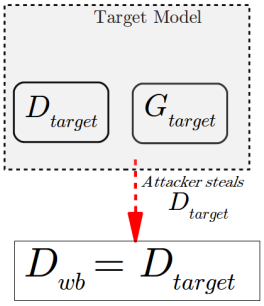



**Figure 6.1:** High-level Outline of the White-Box Attack.

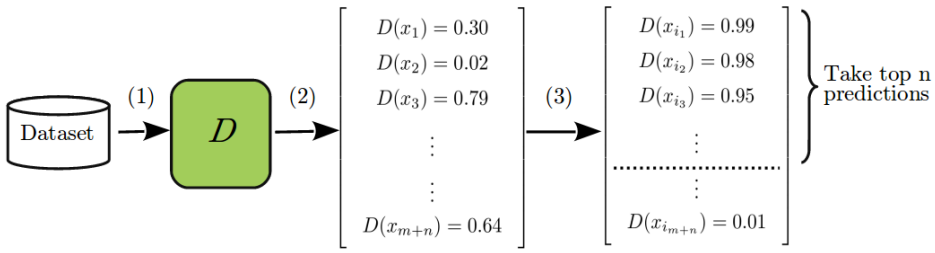



**Figure 6.2:** White-Box Prediction Method: The attacker inputs data-points to the Discriminator $D$ (1), extracts the output probabilities (2), and sorts them (3).

the target model, where $n$ is the size of the training set, and $m$ is the number of data-points that do not belong to the training set.

The target model has been trained to generate samples that resemble the training set samples. $\mathcal{A}_{wb}$ creates a local copy of $D_{target}$, which we refer to as $D_{wb}$. Then, as shown in Figure 6.2, $\mathcal{A}_{wb}$ inputs all samples $X = \{x_1, \ldots, x_{m+n}\}$ into $D_{wb}$, which outputs the resulting probability vector $\mathbf{p} = [D_{wb}(x_1), \ldots, D_{wb}(x_{m+n})]$. If the target model overfitted on the training data, $D_{wb}$ will place a higher confidence value on samples that were part of the training set. $\mathcal{A}_{wb}$ sorts their predictions, $\mathbf{p}$, in descending order and takes the samples associated with the largest $n$ probabilities as predictions for members of the training set.

Note that the attacker does not need to train a model; rather, it relies on internal access to the target model, from which the attack can be launched.

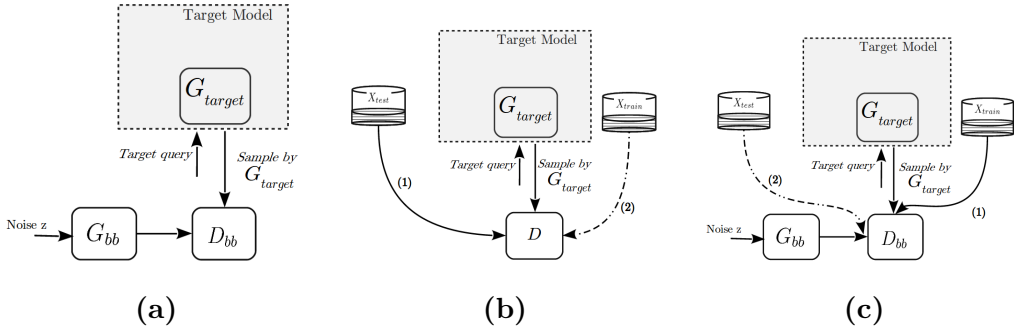



**Figure 6.3:** High-level overview of the (a) black-box attack with no auxiliary knowledge, and (b) *Discriminative* and (c) *Generative* black-box attack with limited auxiliary attacker knowledge.

### 6.1.3 Black-box attack with no auxiliary knowledge

In the black-box setting, we assume that the attacker $\mathcal{A}_{bb}$ does not have access to the target model parameters. Thus, $\mathcal{A}_{bb}$ cannot directly steal the discriminator model from the target as in the white-box attack. Furthermore, while in the white-box attack we restrict the target model to be a GAN, here we do not, and the target model may not have an associated discriminative model (as with VAEs). Again, to evaluate the attack, we assume the attacker has access to a dataset, $X = \{x_1, \ldots, x_{m+n}\}$, which they suspect contains datapoints used to train the target model, where $n$ is the size of the training set. However, the attacker has no knowledge of how the training set was constructed from $X$, thus, they do no have access to the true labels of samples from the dataset and so cannot train a model using a discriminative approach. Instead, $\mathcal{A}_{bb}$ trains a GAN in order to re-create the target model locally and, in the process, creates a discriminator $D_{bb}$, which detects overfitting in the generative target model $G_{target}$. We illustrate the attack in Figure 6.3(a).

More specifically, $\mathcal{A}_{bb}$ locally trains a GAN ($G_{bb}$, $D_{bb}$) using queries from the target, i.e., $\mathcal{A}_{bb}$ trains the local GAN on samples generated by $G_{target}$. Note that as the black-box attack depends only on samples generated by the target model, $G_{target}$ can be any generative model. We assume $\mathcal{A}_{bb}$ has neither knowledge nor control over the source of randomness used to generate the

samples generated by $G_{bb}$. After the GAN has been trained, the attack proceeds to the white-box setting, i.e., $\mathcal{A}_{bb}$ inputs data-points $X$ into $D_{bb}$, sorts the resulting probabilities, and takes the largest $n$ points as predictions for the training set (as shown in Figure 6.2).

### 6.1.4 Black-box attack with limited auxiliary knowledge

In the black-box attack presented above, we assume that $\mathcal{A}_{bb}$ has no additional knowledge about subsets of members of the dataset. However, we also study the case where an attacker could leverage limited additional side information about the training set. This is a realistic setting, which has been considered extensively in the literature. For instance, social graph knowledge has been used to de-anonymize social networks [156]. Overall, auxiliary/incomplete knowledge of sensitive datasets is a common assumption in literature [175, 108].

Access to side information about the training set means that the attacker can "augment" the black-box attack. We consider two attack settings: a generative and a discriminative one. In both settings, we consider a scenario where the attacker has incomplete knowledge of members of the test dataset, the training dataset, or both.

**Discriminative setting.** We consider an attacker that trains a simple discriminative model to infer membership of the training set, as illustrated in Figure 6.3(b). This is feasible since the attacker now has access to membership binary labels, i.e., whether data points belong to the training set or not. Thus, they do not need to train a generative model to detect overfitting. Within this setting, we consider two scenarios:

(1) The attacker has limited auxiliary knowledge of samples that *were not* used to train the target model.

(2) The attacker has limited auxiliary knowledge of *both* training set and test set samples.

In both cases, the general method of attack is the same: an attacker trains a

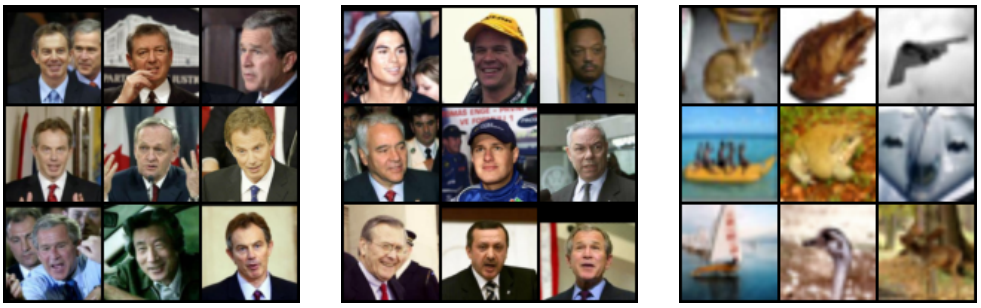
local model to detect overfitting in the target model. In (1), the discriminator, $D$, is fed samples from this auxiliary set, labeled as fake samples, and samples generated by the target model, labeled as real samples. If the target model overfits the training set, $D$ will learn to discriminate between training and non-training samples. In (2), $D$ is fed both target generated samples and the auxiliary training samples, labeled as real samples, and samples from the auxiliary test set, labeled as fake samples. Once the attacker has trained a discriminator, the attack again proceeds as described in Figure 6.2. Note that we have to consider that the attacker knows some test samples (i.e., fake samples) in order to properly train a binary discriminator.

**Generative setting.** We also consider a generative attack, as outlined in Figure 6.3(c), again, as per two scenarios:

(1) The attacker has limited auxiliary knowledge of samples that *were* used to train the target model.

(2) The attacker has limited auxiliary knowledge of *both* training set and test set samples.

With both, the attacker trains a local model – specifically, a GAN – that aims to detect overfitting in the target model. In (1), the discriminator of the attacker GAN, $D_{bb}$, is trained using samples generated by $G_{bb}$, labeled as fake samples, and both samples from the auxiliary training set and target generated samples, labeled as real samples. Intuitively, we expect the attacker model to be stronger at recognizing overfitting in the target model, if it has auxiliary knowledge of samples on which it was originally trained. In (2), $D_{bb}$ is trained on samples generated by $G_{bb}$ and samples from auxiliary set of test ones, labeled as fake samples, and samples generated by the target model and samples from the auxiliary training set, labeled as real ones. The attacker GAN is trained to learn to discriminate between training and non-training samples directly. Again, once the attacker has trained their model, data-points from $X$ are fed into $D_{bb}$, and their predictions are sorted as described in Figure 6.2.



**(a)** LFW, top ten classes

**(b)** LFW, random 10%

**(c)** CIFAR-10, random 10%

**Figure 6.4:** Real samples.

## 6.2 Evaluation

In this section, we present an experimental evaluation of the attacks described above.

### 6.2.1 Experimental setup

**Testbed.** Experiments are performed using PyTorch[1] on a workstation running Ubuntu Server 16.04 LTS, equipped with a 3.4GHz CPU i7-6800K, 32GB RAM, and an NVIDIA Titan X GPU card.

**Settings.** For white-box attacks, we measure membership inference accuracy at successive epochs of training the target model, where one epoch corresponds to one round of training on all training set inputs.[2] For black-box attacks, we fix the target model and measure membership inference accuracy at successive training steps of the attacker model, where one training step is defined as one iteration of training on a mini-batch of inputs. The attacker model is trained using soft and noisy labels as suggested in [183], i.e., we replace labels with random numbers in $[0.7, 1.2]$ for real samples, and random values in $[0.0, 0.3]$ for fake samples. Also, we occasionally flip the labels when training the discriminator. These modifications to the GAN have been shown to stabilize training in practice [52].

[1] `https://github.com/pytorch/pytorch`

[2] We update model weights after training on mini-batches of 32 samples.

**Datasets.** We start by testing our attacks on two popular machine learning datasets, namely, Labeled Faces in the Wild (LFW) [102] and CIFAR-10 [119], then, in Section 6.2.7, we present a case-study evaluation on a medical image dataset, i.e., the diabetic retinopathy (DR) dataset [110]. LFW includes 13,233 images of faces collected from the Web, while CIFAR-10 consists of 60,000 32x32 color images in 10 classes, with 6,000 images per class. For both of them, we randomly choose 10% of the dataset for the training set. Note that LFW is *unbalanced*, i.e., some people appear in multiple images, while others only appear once. We also perform experiments so that the training set is chosen to include the ten most popular classes of people in terms of number of images they appear in, which amounts to 12.2% of the LFW dataset. Intuitively, we expect that models trained on the top ten classes will overfit more than the same models trained on random 10% subsets, as we are training on a more homogeneous set of images. Figure 6.4 shows real samples from LFW and CIFAR-10.

Finally, the DR dataset consists of 88,702 high-resolution retina images taken under a variety of image conditions. From DR, we select images that labeled as having moderate to proliferate diabetic retinopathy presence, and use them to train the generative target model.

**Models.** Since the introduction of GANs [87], a few variants have been proposed to improve training stability and sample quality. In particular, deep convolutional generative adversarial networks (DCGANs) [176] combine the GAN training process with convolutional neural networks (CNNs). CNNs are considered the state-of-the-art for a range image recognition tasks and, by combining CNNs with the GAN training processes, DCGANs perform well at unsupervised learning tasks such as generating complex representations of objects and faces [176].

GANs have also been combined with VAEs [122], by collapsing the generator (of the GAN) and decoder (of the VAE) into one, the model uses learned feature representations in the GAN discriminator as the reconstructive error

term in the VAE. Experimentally, it has been shown that combining the DCGAN architecture with a VAE yields more realistic generated samples [164]. More recently, Boundary Equilibrium GAN (BEGAN) [25] have been proposed as an approximate measure of convergence. Loss terms in GAN training do not correlate with sample quality, making it difficult for a practitioner to decide when to stop training. This decision is usually performed by visually inspecting generated samples. BEGAN proposes a new method for training GANs by changing the loss function. The discriminator is an autoencoder and the loss is a function of the quality of reconstruction achieved by the discriminator on both generated and real samples. BEGAN produces realistic samples [25], and is simpler to train since loss convergence and sample quality is linked with one another.

We evaluate our attacks using, as the target model, DCGAN, DCGAN+VAE, and BEGAN, while fixing DCGAN as the attacker model. This choice of models is supported by recent work by Lucic et al. [135], who show that no other GAN model performs significantly better than our choices. Furthermore, they show VAE models perform significantly worse than any GAN variant.

### 6.2.2 Naïve approaches

We begin our evaluation with a naïve Euclidean distance based attack, which further motivates the use of more sophisticated machine learning techniques. Given a sample generated by a target model, the attacker computes the Euclidean distance between the generated sample and every real sample in the dataset. Repeating this multiple times for newly generated samples, the attacker computes an average distance from each real sample, sorts the average distances, and takes the smallest $n$ distances (and the associated real samples) as the guess for the training set, where $n$ is the size of the training set.

We perform this attack on a target model (DCGAN) trained on a random 10% subset of CIFAR-10 and a random 10% subset of LFW. Figure 6.5 clearly

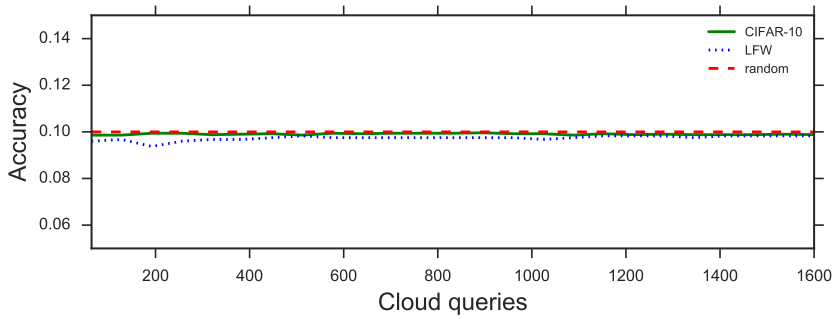



**Figure 6.5:** Euclidean attack results for DCGAN target model trained on a random 10% subset of CIFAR-10 and LFW.

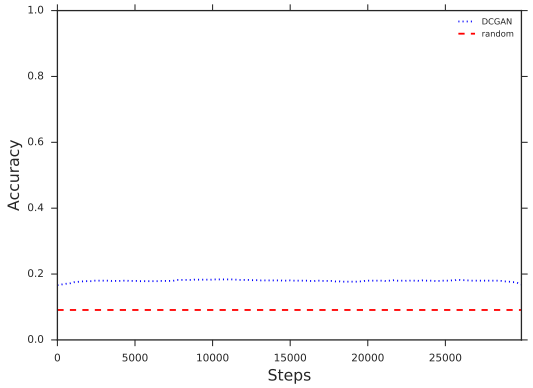
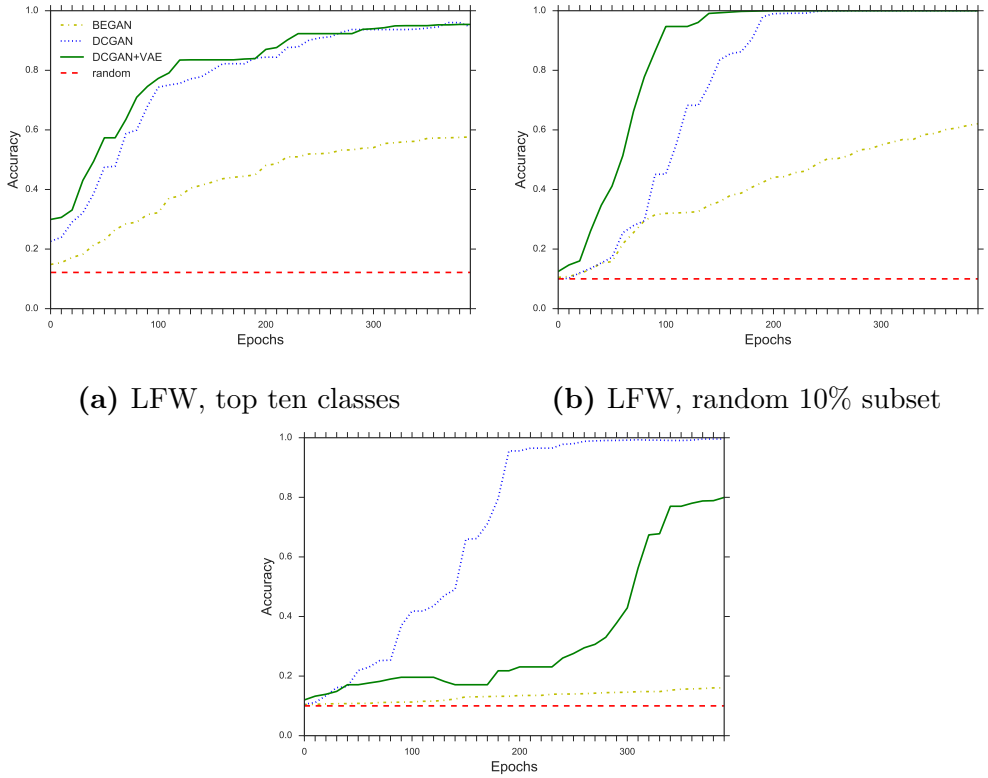



**Figure 6.6:** Black-box attack results with 10% auxiliary attacker training set knowledge used to train a DCGAN *shadow model* for DCGAN target model trained on a random 10% subset of LFW.

shows that this Euclidean attack does not perform better than if the attacker were to randomly guess which real samples were part of the original training set.

We also report on the results of a black-box setting where 10% of training set samples from LFW are used to train a *shadow model*, inspired by the techniques of Shokri et al. [189] – see Figure 6.6. Samples generated by this model are then injected into the attacker model together with the samples generated by the target model. More specifically, at training time, each mini-batch is composed of synthetic samples generated either by the target model or by the shadow model. However, this attack, only yields around 18% of accuracy, with no improvements during training.




**(a)** LFW, top ten classes

**(b)** LFW, random 10% subset

**(c)** CIFAR-10, random 10% subset

**Figure 6.7:** Accuracy of white-box attack with different datasets and training sets.

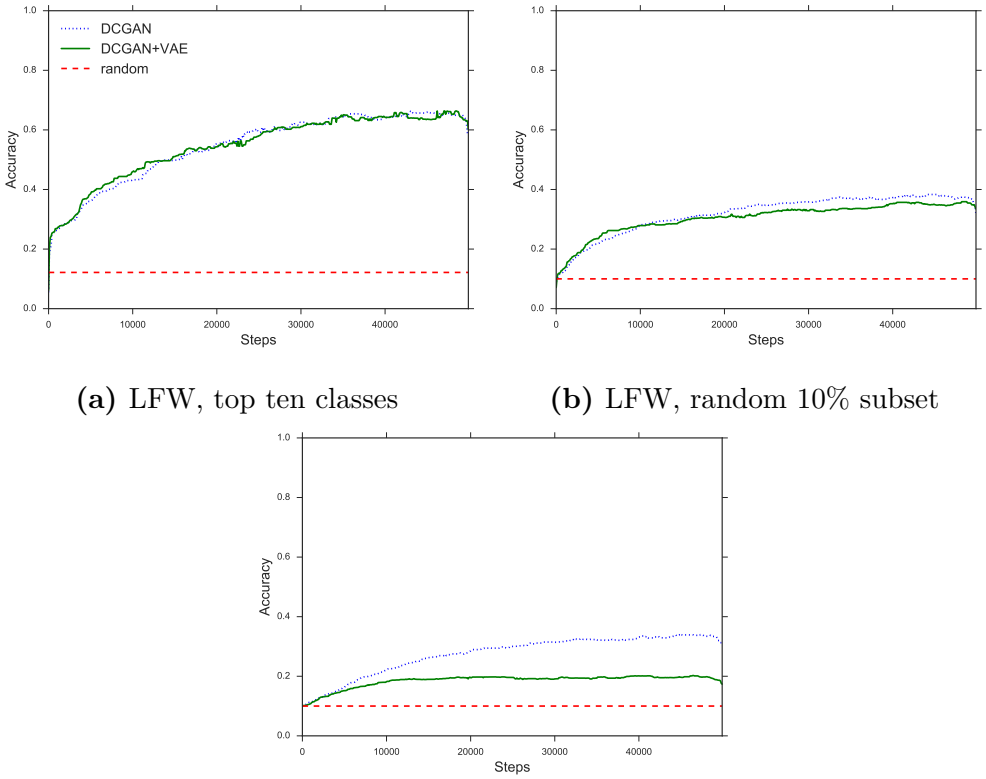
### 6.2.3 White-box attack

We now present the results of our evaluation of the white-box attack described in Section 6.1.2 on LFW and CIFAR-10. For the LFW dataset, we build the training set either as a random 10% subset of the dataset or the top ten classes. For CIFAR-10, the training set is a random 10% subset of the dataset. The target models we implement are DCGAN, DCGAN+VAE, and BEGAN. In the rest of this section, we will include a baseline in the plots (red dotted line) that corresponds to the success of an attacker randomly guessing which samples belong to the training set. We observe that both DCGAN and DCGAN+VAE are vulnerable to the white-box attack. For DCGAN and DCGAN+VAE target models trained for 100 epochs, the attacker infers training set membership with 80% accuracy, and for models trained for 400 epochs – with 98% and 97% accuracy, respectively. The BEGAN target model does

overfit, although to a lesser extent: after 400 epochs, an attacker with white-box access to the BEGAN target model can infer membership of the training set with 60% accuracy. In Figure 6.7(b), we report the results of white-box attacks against a target model trained on a random 10% subset of the LFW dataset. Similar to Figure 6.7(a), both DCGAN and DCGAN+VAE are vulnerable: when these are trained for 250 epochs, an attacker can achieve perfect training set membership inference. BEGAN performs similar to the top ten classes white-box experiment, achieving 62% accuracy after 400 epochs. Finally, Figure 6.7(c) plots the accuracy of the white-box attack against a target model trained on a random 10% subset of CIFAR-10.

For DCGAN, results are similar to DCGAN on LFW, with perfect training set membership inference after 400 epochs. However, DCGAN+VAE does not leak information (does not overfit) until around 250 epochs, where accuracy remains relatively steady, at 10-20%. Instead, after 250 epochs, the model overfits, with accuracy reaching 80% by 400 epochs. BEGAN, while producing quality samples, does not overfit, with final training set membership inference accuracy of 19%, i.e., only 9% better than random guess. Due to the limited accuracy of BEGAN in comparison to other models, we discard it as a target model for black-box attacks as it does not seem to be vulnerable to membership inference attacks. Note that GAN models need to be trained for hundreds of epochs before reaching good samples quality. Indeed, the original DCGAN/BEGAN papers report 2x and 1.5x the number of network updates (when adjusted for training set size) as our white-box attack, to train DCGAN and BEGAN, respectively.

In summary, we conclude that white-box attacks infer the training set with up to perfect accuracy when DCGAN and DCGAN+VAE are the target models. On the other hand, BEGAN is less vulnerable to white-box attacks, with up to 62% accuracy.




**(a)** LFW, top ten classes

**(b)** LFW, random 10% subset

**(c)** CIFAR-10, random 10% subset

**Figure 6.8:** Accuracy of black-box attack on different datasets and training sets.

### 6.2.4 Black-box attack with no auxiliary knowledge

Next, we present the results of the black-box attacks (cf. Section 6.1.3) on LFW and CIFAR-10. We assume the attacker has no knowledge of the training or test sets other than the size of the original training set. Once again, for LFW, the training set is either a random 10% subset of the dataset or the top ten classes, while, for CIFAR-10, the training set is always a random 10% subset of the dataset. The target models we implement are DCGAN and DCGAN+VAE (fixed at epoch 400), and the attacker model uses DCGAN. Figure 6.8(a) plots the results of a black-box attack against a target model trained on the top ten classes of the LFW dataset. After training the attacker model on target queries, the attack achieves 63% training set membership inference accuracy for both DCGAN and DCGAN+VAE target models. Surprisingly, the attack performs equally well when the target model differs

from the attack model as when the target and attack model share the same architecture. This highlights the fact that the attacker does not need to have knowledge of the target model architecture in order to perform the attack.

In Figure 6.8(b), the results are with respect to a target model trained on a random 10% subset of the LFW dataset. Once again, we find that DCGAN and DCGAN+VAE target models are equally vulnerable to a black-box attack. An attacker with no auxiliary information of the training set can still expect to perform membership inference with 40% (38%) accuracy for the DCGAN (DCGAN+VAE) target model.

Finally, Figure 6.8(c) plots the accuracy of a black-box attack against a target model trained on a random 10% subset of the CIFAR-10 dataset. For the DCGAN+VAE target model, accuracy reaches 20% after 1,000 training steps and stays flat. For the DCGAN target model, the attacker can infer training set membership with 37% accuracy, with accuracy improving steadily throughout the attacker model training process.

We observe that the difference in attack success between the DCGAN and DCGAN+VAE target models with CIFAR-10 and the similar success of the two models with LFW occur in both white-box and black-box attacks. As expected, the best results are obtained when the attacker and target model have the same architecture. However, the attack does not overwhelmingly suffer under differing architectures. In fact, in LFW experiments there is a negligible difference in attack success, and, in the CIFAR-10 black-box experiments, the difference in accuracy is approximately 17%.

In summary, we conclude that our black-box attacks are less successful, compared to white-box attack, in inferring membership, but perform similarly against different target model architectures.

### 6.2.5 Black-box attack with limited auxiliary knowledge

As discussed in Section 6.1.4, we also consider black-box attacks where the attacker has some limited auxiliary knowledge of the dataset. We now present the results of these attacks on random 10% subsets of LFW and CIFAR-10

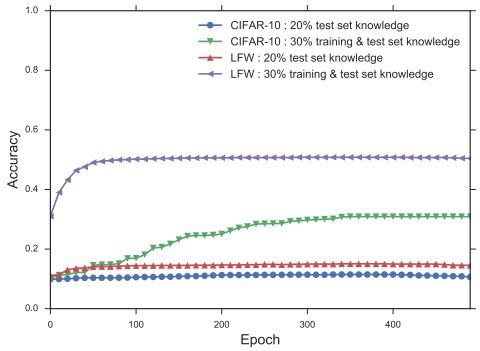


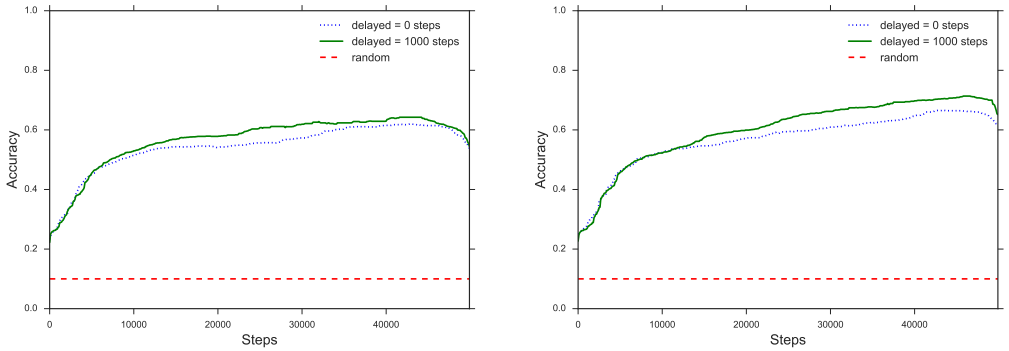

**Figure 6.9:** Membership inference accuracy using a discriminative model, when the attacker has knowledge of (i) 20% of the test set, or (ii) 30% of both training and test sets. In (i), randomly guessing the training set corresponds to 14% accuracy, in (ii), to 12% accuracy.

with DCGAN attacker and target models (fixed at epoch 400).

We consider different scenarios where the attacker has knowledge of 20–30% of the training set, 20-30% of the test set, or both. Nonetheless, the total number of samples of which the attacker has knowledge is quite modest. For LFW, 20% of the random 10% training set corresponds to 264 out of 1,323 images, 20% of the test set to 2,382 out of 11,910 images, whereas, for CIFAR-10, 20% of the random 10% training set amounts to 1,200 out of 6,000 images, and 20% of the test set to 10,000 out of 50,000 images. An attacker with auxiliary information of the training and test set has access to labels, and therefore may not need to train a generative model to perform a membership inference attack on a generative model. We also show that, while the attacker can train a discriminative model to perform membership inference, such an approach produces worse results than the generative method.

If an attacker has access to true labels within the dataset, they can train a discriminative model on these samples in order to learn to classify training samples correctly. For both LFW and CIFAR-10 DCGAN target models, trained on a random 10% subset of the dataset, we consider two settings:

(i) the attacker has 20% knowledge of the *test* set;




**(a)** DCGAN **(b)** DCGAN+VAE

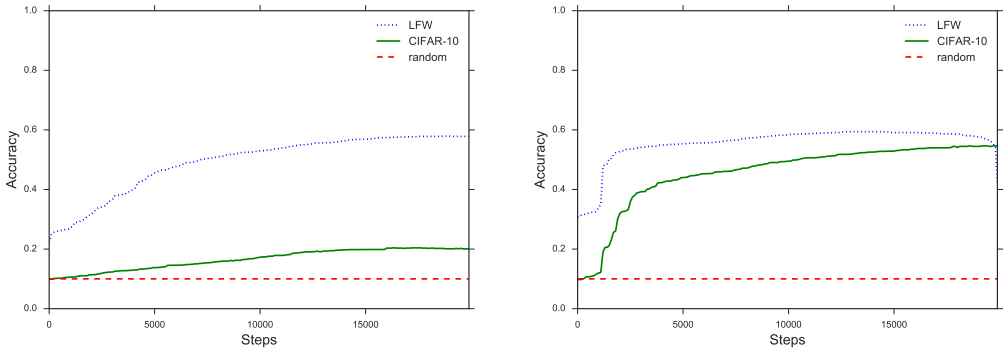
**Figure 6.10:** Black-box attack results with 20% attacker training set knowledge for DCGAN/DCGAN+VAE target models, trained on a random 10% subset of LFW, for different delays at which auxiliary knowledge is introduced into the attacker model training.

(ii) the attacker has 30% knowledge of both the training and test set.

We use the discriminator from DCGAN as the discriminative model trained by the attacker. In (i), we pass test set samples to the discriminator labeled as fake samples, and target generated ones labeled as real ones. In (ii), we pass test set samples to the discriminator labeled as fake ones, and target generated and training set samples labeled as real ones.

In Figure 6.9, we plot the accuracy results for both settings, showing that the attack fails with both datasets when the attacker has only test set knowledge, performing no better than random guessing. Whereas, if the attacker has both training and test knowledge, with LFW, the attacker achieves 50% accuracy, while, for CIFAR-10, accuracy reaches 33%. Note that this approach does not improve on CIFAR-10 black-box results with no auxiliary knowledge, and only marginally improves on LFW results. As a result, we also experiment with generative approaches to black-box attacks with auxiliary attacker knowledge, as discussed next.

We consider the same set of experiments with similar settings for attacker knowledge as in the discriminative approach; the only difference is that in setting (i) we now assume the attacker has 20% knowledge of the training set




**(a)** **(b)**

**Figure 6.11:** Black-box results when the attacker has (a) knowledge of 20% of the training set or (b) 30% of the training set and test set. The training set is a random 10% subset of the LFW or CIFAR-10 dataset, and the target model is fixed as DCGAN.

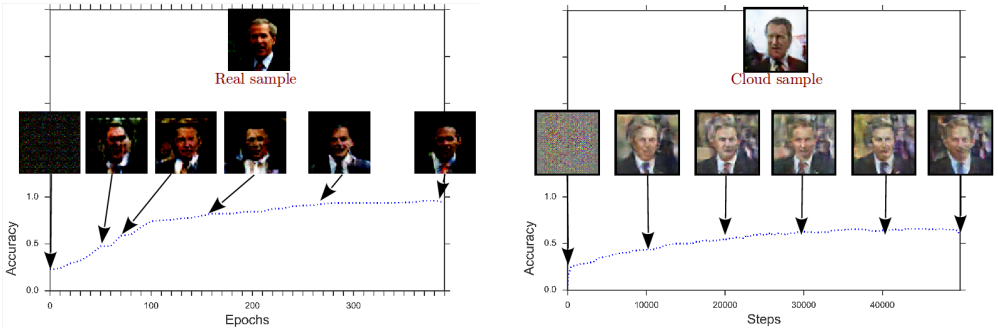
rather than the test set. We use DCGAN as the generative attacker model. Specifically, we consider two settings in which:

(1) the attacker has 20% knowledge of the *training* set,

(2) the attacker has 30% knowledge of both the training and test set.

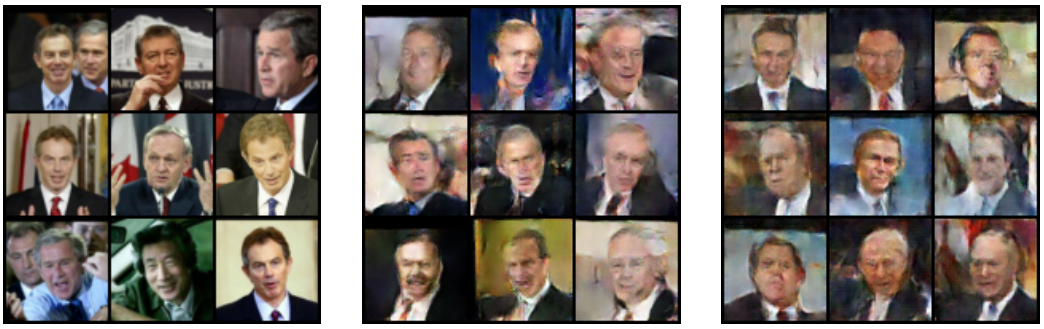
In all the experiments, we introduce a delay of 1000 training steps before the attacker model uses the auxiliary attacker knowledge. We found that introducing the auxiliary knowledge early in training process of the attacker model resulted in a weaker discriminator – see Figure 6.10. Figure 6.11(a) shows results for setting (1): clearly, there is a substantial increase in accuracy for the LFW dataset, from 40% attack accuracy to nearly 60%. However, there is no increase in accuracy for the CIFAR-10 dataset. Thus, we conclude that setting (1) does not generalize. Figure 6.11(b) shows results for setting (2); for both LFW and CIFAR-10 there is a substantial improvement in accuracy. Accuracy for the LFW experiment increases from 40% (with no auxiliary attacker knowledge) to 60%, while, for CIFAR-10, from 37% to 58%.

Thus, we conclude that, even a small amount of auxiliary attacker knowledge can lead to greatly improving membership inference attacks.




**(a)** White-box attack. **(b)** Black-box attack.

**Figure 6.12:** Accuracy curves and samples at different stages of training on top ten classes from the LFW dataset, showing a clear correlation between higher accuracy and better sample quality.



**(a)** Real samples **(b)** Target samples **(c)** Attacker samples

**Figure 6.13:** Various samples from the real dataset, target model, and black-box attack using the DCGAN target model on LFW, top ten classes.

### 6.2.6 Analysis

Aiming to better understand the relationship between membership inference and training performance, we report, in Figure 6.12, the attack accuracy and samples generated at different training stages by the target DCGAN generator in the white-box attack  Figure 6.12(a)) and the attacker DCGAN generator in the black-box attack  Figure 6.12(b)) on the top ten classes from the LFW dataset. The plots demonstrate that accuracy correlates well with the visual quality of the generated samples. In particular, samples generated by the target yield a better visual quality than the ones generated by the attacker generator during the black-box attack, and this results in higher member-

ship inference accuracies. Overall, the samples generated by both attacks at later stages look visually pleasant, and fairly similar to the original ones. Our attacks have been evaluated on datasets that consist of complex representations of faces (LFW) and objects (CIFAR-10). As shown in Figure 6.13(a), real samples from LFW contain rich details both in the foreground and background. We do not observe any large deviations in images within datasets, excluding the possibility that the attack performs well due to some training samples being more easily learned by the model, and so predicting with higher confidence. Learning the distribution of such images is a challenging task compared to simple datasets such as MNIST. In fact, our black-box attack is able to generate realistic samples (see differences between the target model samples in Figure 6.13(b) and the attacker samples in Figure 6.13(c)).

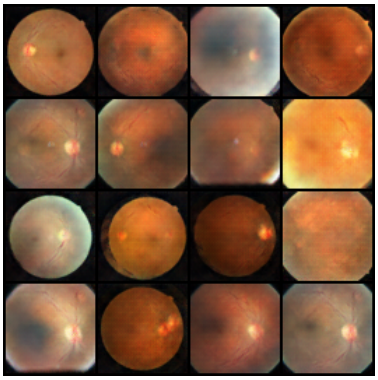
### 6.2.7 Evaluation on Diabetic Retinopathy dataset

Finally, we present a case study of our attacks on the Diabetic Retinopathy (DR) dataset, which consists of high-resolution retina images, with an integer label assigning a score of how much the participant suffers from diabetic retinopathy. Diabetic retinopathy is a leading cause of blindness in the developed world, with detection currently performed manually by highly skilled clinicians. The machine learning competition site `kaggle.com` has evaluated proposals for automated detection of diabetic retinopathy, and submissions have demonstrated high accuracies, thus removing the need for the time-consuming process of manual detection. We choose this additional dataset since the generation of synthetic medical images through generative models is nowadays a powerful method to produce large numbers of high-quality sample data on which machine learning models can be trained. Thus, our attacks can raise serious privacy concerns, in practice, in sensitive settings like those involving medical data.

As discussed in Section 6.2.1, the DR dataset consists of 88,702 high-resolution retina images under various imaging conditions. Each image has an associated integer label assigning how present diabetic retinopathy is within

**(a)** Real sample with no presence of diabetic retinopathy.

**(b)** Real sample with high presence of diabetic retinopathy.

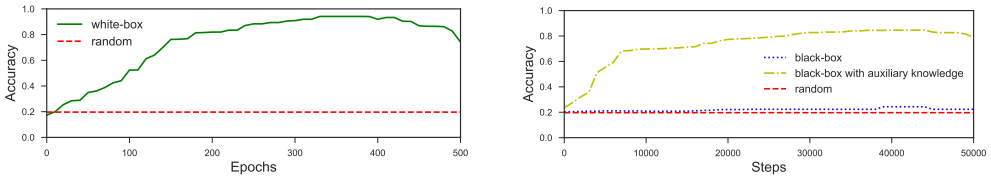


**(c)** Selection of target generated samples classified with high confidence as belonging to the training set by both white-box and black-box attacks.

**Figure 6.14:** Real and generated diabetic retinopathy dataset samples.

the retina, from 0 to 4. We train the generative target model on images with labels 2, 3 and 4, i.e., images with mild to severe cases of diabetic retinopathy. These make up 19.7% of the dataset. Figure 6.14 shows real and target generated samples of retina images.

The results of the white-box attack are reported in Figure 6.15(a): the attack is overwhelmingly successful, nearing 100% accuracy at 350 training epochs. Then, Figure 6.15(b) shows the black-box attacks results, when an attacker has no auxiliary knowledge, and when the attacker has 30% training




**(a)** White-box attack. **(b)** Black-box attack.

**Figure 6.15:** Accuracy curves of attacks against a DCGAN target model on the Diabetic Retinopathy dataset.

| **Attack** | **LFW** | **CIFAR-10** | **DR** |
|---|---|---|---|
| White-box | 100% | 100% | 95% |
| Black-box with no knowledge | 40% | 37% | 22% |
| Black-box with limited knowledge | 60% | 58% | 81% |

**Table 6.1:** Accuracy of the best attacks on random 10% training set for LFW and CIFAR-10, and for the diabetic retinopathy (DR) experiments.

and test set auxiliary knowledge. A no-knowledge black-box attack does not perform very well, while, with some auxiliary knowledge, it approaches the accuracy of the white-box attack, peaking at over 80% after 35K training steps.

## 6.3 Discussion

Overall, our analysis shows that state-of-the-art generative models are vulnerable against membership inference attacks. In Table 6.1, we summarize the best accuracy results obtained for experiments on random 10% training sets (LFW, CIFAR-10) and the diabetic retinopathy (DR) dataset experiments. We note that, for white-box attacks, the attacker successfully infers the training set with 100% accuracy on both the LFW and CIFAR-10 datasets, and 95% accuracy for DR dataset. Accuracy drops to 40% on LFW, 37% on CIFAR-10 and 22% on DR for black-box attacks with no auxiliary knowledge, however, even with a small amount of auxiliary knowledge, the attacker boost performance up to 60% on LFW, 58% on CIFAR-10 and 81% on DR. (A ran-

dom guess corresponds to 10% accuracy on LFW and CIFAR-10, and 20% on DR.) Furthermore, our attacks are robust against different target model architectures.

In this section, we first discuss the computational costs of our attacks, then, we measure the sensitivity of the attacks to training set size and prediction ordering, and study their robustness against possible defenses.

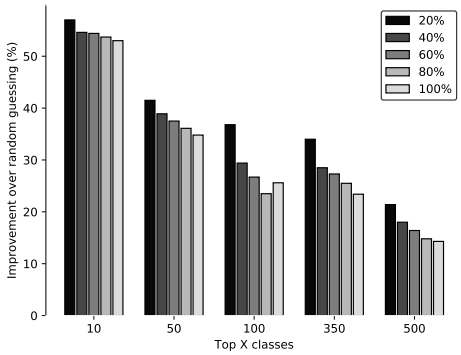
### 6.3.1 Cost of the attacks

Finally, we quantify the cost of the attacks in terms of computational and time overhead, and estimate monetary costs.

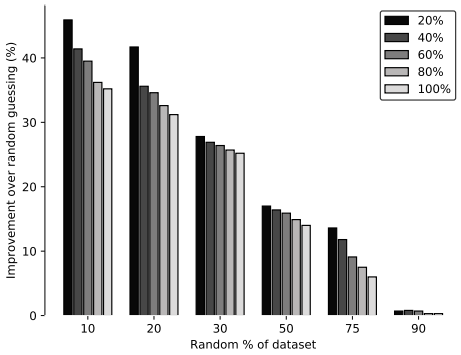
To perform the attacks, the attacker needs a GPU, which can be obtained for a cost in the order of $100. The attacks have minimal running time overheads: for the white-box attack, complexity is negligible as we only query a pre-trained target model to steal discriminator model parameters, whereas, for black-box, one step of training the attacker model takes 0.05 seconds in our testbed. Black-box attacks with no auxiliary attacker knowledge yield the best results after 50,000 training steps, therefore, an attacker can expect best results after approximately 42 minutes with $32 \times 50{,}000$ queries to the target model (since we define one training step as one mini-batch iteration, with 32 inputs per mini-batch). For attacks with auxiliary knowledge, the best results are reached after 15,000 training steps, thus, approximately 13 minutes.

We also estimate monetary cost based on current discriminative MLaaS pricing structures from Google.[3] At a cost of $1.50 per 1,000 target queries, after an initial 1,000 free monthly queries, the black-box attack with no auxiliary knowledge would cost $2,352, while the black-box attack with auxiliary knowledge $672. Therefore, we consider our attacks to have minimal costs, especially considering the potential severity of the information leakage they enable.

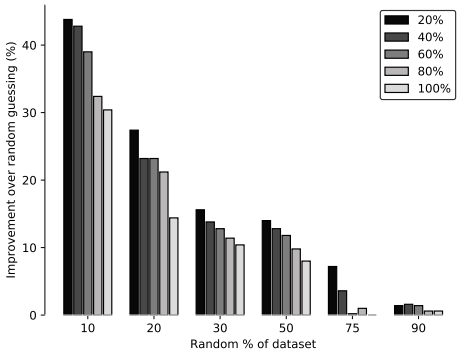
[3] `https://cloud.google.com/vision/pricing`




**(a)** LFW Top X classes




**(b)** LFW, random X% subset




**(c)** CIFAR-10 random X% subset

**Figure 6.16:** Improvements over random guessing, in a black-box attack, as we vary the size of the training set, and consider smaller subsets for training set predictions.

### 6.3.2 Sensitivity to training set size and prediction ordering

Aiming to measure the dependency between attack performance and the training set size, we also experiment with varying training set sizes in the DCGAN target and attacker model setting. Figure 6.16 shows how the improvement of the attack degrades as the relative size of the training set increases. Note that we only include black-box attack results, as all white-box attacks achieve almost 100% accuracy *regardless of training set size.* Overall, we find that there is a commonality in the experiments: black-box attacks on 10% of the dataset achieve an improvement of 40–55%, and, as we increase the number of data-points used to train the target model, the attack has smaller and smaller improvements over random guessing. The largest increases are in

the setting of Figure 6.16(a), where data-points are more homogeneous and so overfitting effects are compounded. When the training set is 90% of the total dataset used in the evaluation of the attack, the attack has negligible improvements over random guessing.

We believe that this is due to: (1) the larger number of training data-points yields a well-fitted model that does not leak information about training records, and (2) a small number of data-points within the training set do not leak information, therefore, as we increase the size of the training set, the inability to capture these records becomes more costly, resulting in smaller improvements in attack performance. If (1) were true, we would see smaller improvements for larger training sets, regardless of the total size of the dataset; however, experiments on both LFW and CIFAR-10, which consist of different training sizes, report similar improvements over random guessing. Additionally, white-box attacks are not affected by increasing the training set size, which would be the case if the model did not overfit and thus leak information about training records. Hence, we believe that a small number of training records are inherently difficult to capture, and so improvements over random guessing for larger training set sizes are more difficult to achieve since the majority of samples are used to train the target model. We also examine the attack sensitivity to the ordering of the data-point predictions. So far, the *only* prior knowledge the attacker has is the approximate size of the training set. If there is a clear ordering of data-points predictions, with training records sitting at the top of the ordering, and non-training records lower down, an attacker can use this information to identify training records without side knowledge of training set size. They can simply place a confidence score relative to where in the ordering a data-point predictions sits. Figure 6.16 shows, for varying training set sizes, how many training records lie in the top 20%, 40%, 60%, 80%, and 100% of the guessed training set. We observe that, in all experimental settings, accuracy for the top 20% is highest, with scores decreasing as the attacker considers a larger number of data-points as candidates for the

training set. Thus, training to non-training samples follow a structured ordering in the attacker's predictions, which can be exploited to infer membership when the attacker has *no knowledge* of the original training set size by setting a threshold on the minimum confidence of a training point.

### 6.3.3 Defenses

Possible defense strategies against membership inference discussed in [189], such as restricting the prediction vector to the top $k$ classes, coarsening and increasing the entropy of the prediction vector, are not well suited to our attacks, since generative models do not output prediction vectors. However, regularization techniques could possibly be applied to generative models to produce more robust and stable training as well as more diverse and visually pleasant samples.

**Weight Normalization and Dropout.** To this end, we consider two techniques, namely, Weight Normalization [184] and Dropout [194], as possible defense mechanisms and evaluate their impact on our attacks.[4] The former is a re-parameterization of the weights vectors that decouples the length of those weights from their direction, and it is applied to all layers in both generator and discriminator in the target model. Whereas, the latter can be used to prevent overfitting by randomly dropping out (i.e., zeroing) connections between neurons during training. In particular, we apply Dropout, with probability 0.5, to all the layers in the discriminator.

In Figure 6.17, we measure the improvement over random guessing for the white-box attack against the target model trained on LFW using either Weight Normalization or Dropout. We find that Dropout is more effective, with improvements over random guessing ranging from 70% on top 10 classes to 23% on top 500 classes. Weight Normalization only yields improvements of, respectively, 88% and 46%, which are very close to the target model trained with no

[4]Note that we do not compare models with and without Batch Normalization [104], as its inclusion has shown to improve sample quality and is nearly always used in model construction of GANs [176].

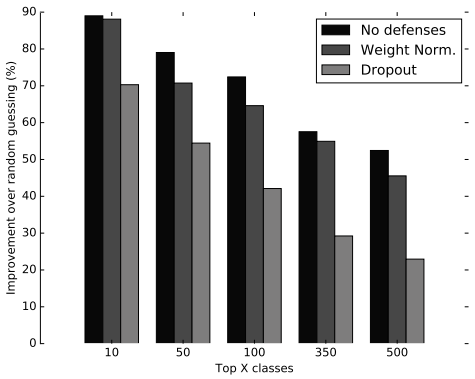



**Figure 6.17:** Improvement over random guessing for Weight Normalization and Dropout defenses against white-box attacks on models trained over different number of classes with LFW.

defenses (resp., 89% and 52%). However, we also find that Weight Normalization often results in training instability (i.e., the discriminator outperforms the generator, or vice-versa), while Dropout significantly slows down the training process, requiring more epochs to get qualitatively plausible samples.

**Using our attacks as defense.** Also note that, our attacks can actually be used as a defense mechanism. The difference in white-box and black-box accuracy provides information about how well the local model approximates the target model, thus, one could use this information to train a target model which cannot be well approximated. Furthermore, similarly to *early-stopping* criteria in model training, one can stop training when visual sample quality is high but white-box attack accuracy is still low.

In our experiments, we also observe the benefits of a more regularized model in increasing the robustness against information leakage in the case of BEGAN. For instance, in white-box attacks on CIFAR-10, BEGAN produces quality samples without overfitting, with membership inference performing only 9% better than random guessing (see Figure 6.7(c)).

**Differentially private GANs.** Finally, we evaluate our attack against a recently proposed technique for $(\varepsilon, \delta)$-Differentially Private GANs [202], where gaussian noise [70] is injected in the discriminator forward pass during train-

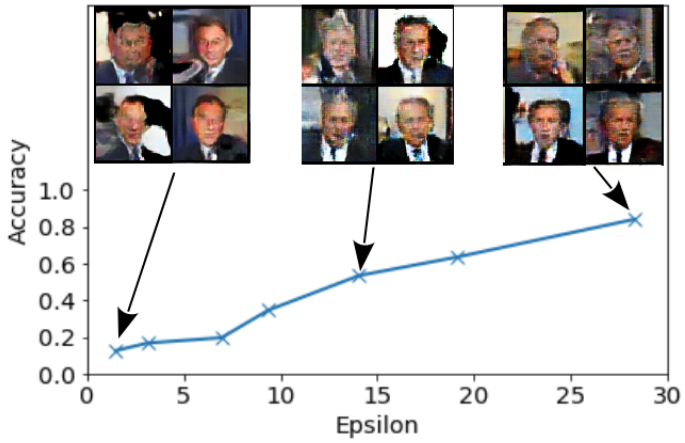



**Figure 6.18:** Accuracy curve and samples for different privacy budgets on top ten classes from the LFW dataset, showing a trade-off between samples quality and privacy guarantees.

ing. Figure 6.18 shows the results of a white-box attack against Differentially Private DCGAN trained on top ten classes for different values of the privacy budget $\varepsilon$ (with $\delta$ set to $10^{-4}$). For all experiments, the target model is trained for 500 epochs and the final privacy budget is computed using moments accountant [1]. The attack does no better than random guessing for $\varepsilon = 1.5$ (first tick in the plot), while accuracy increases up to 85% for $\varepsilon = 28.3$. However, note that acceptable levels of privacy (i.e., values of $\varepsilon < 10$) yield very bad samples quality.

# Chapter 7

# Evaluating Privacy Leakage of Collaborative Learning

In collaborative and federated learning, participants' training data may not be identically distributed. In fact, federated learning is explicitly designed to take advantage of the fact that participants may have private training data that is different from the publicly available data for the same class [143].

In this chapter, we focus on inferring information that is true specifically about certain subsets of the data used to train collaborative learning models. The basic privacy violation in this setting is membership inference. In Chapter 6, we presented some novel results on inferring membership in generative models, but collaborative learning presents interesting new avenues for such inferences.

As discussed in Section 3.2.4, prior work [84, 96, 15] aimed to infer properties that characterize an entire class: for example, given a facial recognition model where one of the classes is Bob, infer what Bob looks like. By contrast, in this work, we aim to infer properties that are true of a specific subset of the training inputs but not of the class as a whole. For instance, when Bob

---

The research presented in this chapter was joint work with a fellow PhD student Congzheng Song. The author equally contributed to the design of the attacks presented in Section 7.1, and lead the work on the experimental evaluation of the attacks on the CSI Corpus and FourSquare datasets in Section 7.2, as well as the evaluation of potential defenses described in Section 7.3.

uses his photos to collaboratively train a gender classifier, we infer that Alice appears in some of the photos. We especially focus on the properties that are *independent* of the class' characteristic features. For example, in the gender classifier example, we infer whether people in Bob's photos wear glasses, even though wearing glasses has no correlation with gender. There is no legitimate reason for a model to leak this information; this is purely an artifact of the collaborative learning process.

A participant's contribution to each iteration of collaborative learning is based on a batch of his training data. We infer *single-batch properties*, i.e., detect that the data in a given batch has the property but other batches do not. We also infer *when a property appears in the training data*. This has potentially serious privacy implications. For instance, we can infer when a certain person starts appearing in a participant's photos or when the participant starts visiting a certain type of doctors. Finally, we infer properties that characterize a participant's entire dataset (but not the entire class), e.g., authorship of the texts used to train a sentiment analysis model.

## 7.1 Inference Attacks

This section describes our inference methodology.

### 7.1.1 Threat model

We assume that $K$ participants (where $K \geq 2$) jointly train a machine learning model using one of the collaborative learning algorithms described in Section 2.3.5. Note that, although we only experiment with joint models trained with SGD, our inference methodology is not specific to SGD. One of the participants is the *adversary*. His goal is to infer information about the training data of another, *target participant* by analyzing periodic updates to the joint model during training. Multi-party ($K > 2$) collaborative learning also involves honest participants who are neither the adversary nor the target. Depending on how collaborative learning is done, their updates or models may be aggregated with those of the target.

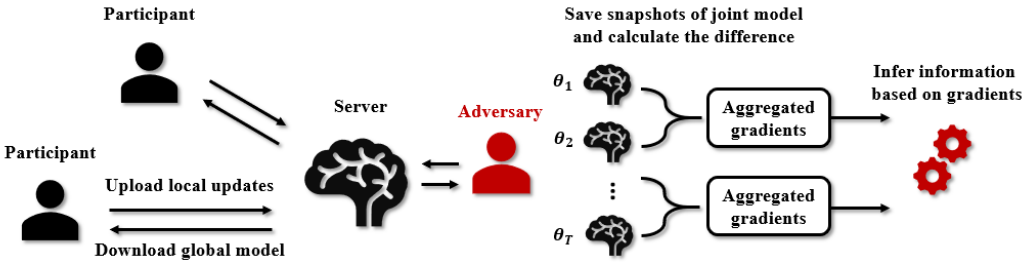



**Figure 7.1:** Overview of inference attacks against collaborative learning.

In the multi-party case, the identities of the participants may not be known to the adversary. If the identities are known but the models are aggregated, the adversary may infer something about the training data but not trace it to a specific participant. In general, tracing requires auxiliary information specific to the leakage. For example, after inferring that photos of a certain person have started appearing in the training data, the adversary may have enough auxiliary and contextual information about the participants to guess which of them has included these photos in the training data. This type of leakage is outside the scope of our work, which focuses solely on the leakage from the collaborative learning process itself.

### 7.1.2 Overview of the attacks

Figure 7.1 gives a high-level overview of our inference attacks. At each iteration $t$ of training, the adversary downloads the current joint model, calculates gradient updates as prescribed by the collaborative learning algorithm, and sends his own updates to the server. The adversary saves the snapshot of the joint model parameters $\theta_t$. The difference between the consecutive snapshots $\Delta\theta_t = \theta_t - \theta_{t-1} = \sum_k \Delta\theta_t^k$ is equal to the aggregated updates from all participants, hence $\Delta\theta_t - \Delta\theta_t^{\text{adv}}$ are the aggregated updates from all participants other than the adversary.

**Leakage from the embedding layer.** For machine learning tasks involving text or location data, where the input space is discrete and sparse, the standard approach is to apply an embedding layer that transforms sparse inputs into a lower-dimensional vector representation. For convenience, we use *word* to

denote discrete tokens: words in the case of text data, specific locations in the case of location data. Let vocabulary $V$ be the set of all words. Each word in the training data is mapped to a word-embedding vector via an embedding matrix $W_{\text{emb}} \in \mathbb{R}^{|V| \times d}$, where $|V|$ is the size of the vocabulary and $d$ is the dimensionality of the embedding.

During training, the embedding matrix is treated as a parameter of the model and optimized collaboratively. The gradient of the embedding layer is sparse with respect to the input words: given a batch of text, the embedding is updated only with the words that appear in the batch. The other words get zero gradients. This difference directly reveals which words occur in the training batches used by the honest participants during collaborative learning.

**Leakage from the gradients.** In deep learning models, the gradients are computed using the back-propagation algorithm which propagates the loss through the entire network from the last to the first layer. Gradients of a given layer are computed using this layer's features and the error from the layer above. In the case of sequential fully connected layers $h_l, h_{l+1}$ ($h_{l+1} = W_l \cdot h_l$, where $W_l$ is the weight matrix), the gradient of error $E$ with respect to $W_l$ is computed as $\frac{\partial E}{\partial W_l} = \frac{\partial E}{\partial h_{l+1}} \cdot h_l$. The gradients of $W_l$ are inner products of the error from the layer above and the features $h_l$. Similarly, for a convolutional layer, the gradients of the weights are convolutions of the error from the layer above and the features $h_l$.

Gradients are based on features, thus observations of the participants' gradient updates can be used to infer the feature values, which are in turn based on these participants' private training data.

### 7.1.3 Membership inference

As explained above, the non-zero gradients of the embedding layer reveal which words appear in a batch. The adversary can use this information to infer membership of a given text or location, i.e., whether this record was part of the training set or not.

Let $V_t$ be the words included in the updates $\Delta\theta_t$. During training, the

**Algorithm 7:** Batch Property Classifier

**Input:** Attacker's auxiliary data $X^{\text{adv}}_{\text{prop}}, X^{\text{adv}}_{\text{nonprop}}$

**Output:** Batch property classifier $f_{\text{prop}}$

$G_{\text{prop}} \leftarrow \emptyset$ ▷ Positive training data for property inference

$G_{\text{nonprop}} \leftarrow \emptyset$ ▷ Negative training data for property inference

**for** $i = 1$ *to* $T$ **do**

  Receive $\theta_t$ from server

  Run **ClientUpdate**$(\theta_t)$

  Sample $B^{\text{adv}}_{\text{prop}} \subset X^{\text{adv}}_{\text{prop}}, B^{\text{adv}}_{\text{nonprop}} \subset X^{\text{adv}}_{\text{nonprop}}$

  Calculate $g_{\text{prop}} = \nabla L(B^{\text{adv}}_{\text{prop}}; \theta_t), g_{\text{nonprop}} = \nabla L(B^{\text{adv}}_{\text{nonprop}}; \theta_t)$

  $G_{\text{prop}} \leftarrow G_{\text{prop}} \cup \{g_{\text{prop}}\}$

  $G_{\text{nonprop}} \leftarrow G_{\text{nonprop}} \cup \{g_{\text{nonprop}}\}$

Label $G_{\text{prop}}$ as positive and $G_{\text{nonprop}}$ as negative

Train a binary classifier $f_{\text{prop}}$ given $G_{\text{prop}}, G_{\text{nonprop}}$

attacker collects a vocabulary sequence $[V_1, \ldots, V_T]$. Given a text record $r$, with words $V_r$, he can test if $V_r \subseteq V_t$, for some $t$ in the vocabulary sequence. If $r$ is in target's dataset, then $V_r$ will be included in at least one vocabulary from the sequence. The adversary can use this to decide whether $r$ was a member or not.

Finally, note that in case of a network initialized with pre-trained word embeddings, and then jointly trained without updating the embedding layer parameters, the adversary would not be able to observe any non-zero gradients for the embedding layers.

### 7.1.4 Passive property inference

We assume that the adversary has auxiliary data consisting of the data points that have the property of interest ($X^{\text{adv}}_{\text{prop}}$) and data points that do not have the property ($X^{\text{adv}}_{\text{nonprop}}$). These data points need to be sampled from the same class as the target participant's data, but otherwise can be unrelated.

The intuition behind this attack is that the attacker can leverage the

snapshots of the global model to generate aggregated updates based on the data with the property and updates based on the data without the property. This produces labeled examples, which enable the adversary to train a binary *batch property classifier* that determines if the observed gradient updates are based on the data with or without the property.

**Batch property classifier.** Algorithm 7 shows how to build a batch property classifier during collaborative training. Given a model snapshot $\theta_t$, calculate the gradients $g_{\text{prop}}$ based on a batch with the property $B^{\text{adv}}_{\text{prop}} \subset X^{\text{adv}}_{\text{prop}}$ and $g_{\text{nonprop}}$ based on a batch without the property $B^{\text{adv}}_{\text{nonprop}} \subset X^{\text{adv}}_{\text{nonprop}}$. Once enough labeled gradients have been collected, train a binary classifier $f_{\text{prop}}$.

For the property inference attacks that exploit the embedding-layer gradients (e.g., the attack on the Yelp dataset in Section 7.2.3), we use a logistic regression classifier.

For all other property inference attacks, we experimented with logistic regression, gradient boosting, and random forests and found that random forests (with 50 trees) performed the best. The input features in this case correspond to the observed gradient updates. The number of the features is thus equal to the model's parameters, which can be very large for a realistic model. To downsample the features representation, we apply the global max pooling operator [86] on the observed gradient updates.

**Property inference.** As collaborative training progresses, the adversary observes gradient updates $g_{\text{obs}} = \Delta\theta_t - \Delta\theta^{\text{adv}}_t$. The basic attack is single-batch inference: the adversary simply feeds the observed gradient updates to the batch property classifier $f_{\text{prop}}$.

This inference attack can be extended from the properties of a single batch to the target's entire training dataset. The batch property classifier $f_{\text{prop}}$ outputs a score in [0,1], indicating the probability that a batch has the property. The adversary can use the average score across all iterations to decide whether the target's entire dataset has the property in question.

This inference attack is *passive*. The adversary observes the gradient

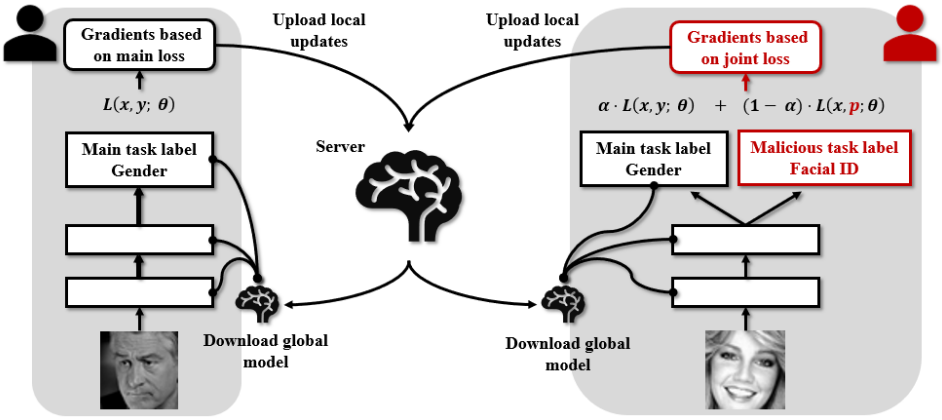



**Figure 7.2:** Active property inference attack.

updates and performs inference without changing anything in the local or global collaborative training procedure.

### 7.1.5 Active property inference

An *active* adversary can perform a more powerful property inference attack by using *multi-task learning*.

The adversary extends his local copy of the main, collaboratively trained model with an augmented property classifier connected to the last layer. This local model is trained simultaneously to perform well on the main task and to recognize batch properties. On the training data where each record has a main label $y$ and a property label $p$, the model's joint loss is calculated as

$$\mathcal{L}_{\text{mt}} = \alpha \cdot \mathcal{L}(x, y; \theta) + (1 - \alpha) \cdot \mathcal{L}(x, p; \theta)$$

During collaborative training, the adversary uploads the updates $\nabla \mathcal{L}_{\text{mt}}$ based on this joint loss. These updates optimize the main model *and* simultaneously learn separable representations for the data with and without the property. As a result, the gradients will be separable, too (e.g., see Figure 7.6 in Section 7.2.5), enabling the adversary to tell if the training data has the property. Figure 7.2 shows an example of active property inference attack with gender classification as the main task and author identification as the inference task.

This adversary is still "honest-but-curious" in the cryptographic parlance. He faithfully follows the collaborative learning protocol and does not submit any malformed messages. The only difference with the passive attack is that this adversary performs additional *local* computations and submits the resulting values into the collaborative learning protocol. Note that the "honest-but-curious" model does not constrain the parties' input values, only their messages.

## 7.2 Experiments

Our experiments encompass a few different machine learning tasks and few datasets, which we review next.

### 7.2.1 Datasets and model architectures

We now describe the datasets, collaborative learning tasks, and adversarial inference tasks used in our experiments—see the summary in Table 7.1. Our choices of hyper-parameters are based on the standard models from the machine learning literature.

**Labeled Faces In the Wild (LFW).** LFW [102] contains 13,233 62x47 RGB face images for 5,749 individuals with different facial-attribute labels such as gender, race, age, hair color, and eyewear.

**FaceScrub.** This dataset [159] contains 76,541 50x50 RGB images for 530 individuals with the gender label: 52.5% are labeled as male, the rest as female. For our experiments, we select a subset of 100 individuals with the most images, for a total of 18,809 images.

On both LFW and FaceScrub, the collaborative models are convolutional neural networks (CNN) with three spatial convolution layers with 32, 64, and 128 filters, kernel size set to $(3, 3)$, and max pooling layers with pooling size set to 2, followed by two fully connected layers of size 256 and 2. We use rectified linear unit (ReLU) as the activation function for all layers. Batch size is 32 (except in the experiments where we vary it), SGD learning rate is 0.01.

| Dataset | #Records | Main Tasks | Inferences |
|---|---|---|---|
| **LFW** | 13.2k | Gender/Smile/Age Eyewear/Race/Hair | Race/Eyewear |
| **FaceScrub** | 18.8k | Gender | Identity |
| **PIPA** | 18.0k | Age | Gender |
| **CSI** | 1.4k | Sentiment | Membership, Region/Gender/Veracity |
| **FourSquare** | 15.5k | Gender | Membership |
| **Yelp-health** | 17.9k | Review score | Membership, Doctor specialty |
| **Yelp-author** | 16.2K | Review score | Author |

**Table 7.1:** Datasets and tasks used in our experiments.

**People in Photo Album (PIPA).** PIPA [226] contains over 60,000 photos of 2,000 individuals collected from public Flickr photo albums. Each image includes one or more people and is labeled with attributes such as the number of people and their gender, age, and race. For our experiments, we select a subset of 18,000 images with three or fewer people and scaled the raw images to 128x128.

The collaborative model for PIPA is a VGG-style [191] 10-layer CNN with five convolution blocks. The first two consist of one convolutional layer and max pooling, the next three of two convolutional layers and max pooling. After the block, there are two fully connected layers. Batch size is 32, SGD learning rate is 0.01.

**Yelp-health.** We extract health care-related reviews from the Yelp dataset[1] of 5 million reviews of businesses listed on Yelp, tagged with numeric ratings (1-5) and attributes such as business type and location. Our subset contains 17,938 reviews for 10 types of medical specialists (see the leftmost column of

[1] `https://www.yelp.com/dataset`

Table 7.4).

**Yelp-author.** We also extract a subset of the Yelp dataset with the reviews of the top 10 most prolific reviewers, 16,207 in total.

On both Yelp datasets, the model is a recurrent neural network with a word-embedding layer of dimension 100. Words in a review are mapped to a sequence of word-embedding vectors, which is fed to a gated recurrent unit (GRU [54]) layer that maps it to a sequence of hidden vectors. We add a fully connected classification layer to the last hidden vector of the sequence. SGD learning rate is 0.05.

**FourSquare.** In [217, 218], Yang et al. collect a global dataset of FourSquare location "check-ins" (userID,time,location,activity) from April 2012 to September 2013. For our experiments, we select a subset of 15,548 users who checked in at least 10 different locations in New York City and for whom we know their gender [219]. This yields 528,878 check-ins. The model is a gender classifier, a task previously studied by Pang et al. [165] on similar datasets.

**CLiPS Stylometry Investigation (CSI) Corpus.** This annually expanded dataset [208] contains student-written essays and reviews. The 1,412 reviews are equally split between Truthful/Deceptive or Positive/Negative and labeled with the attributes of the author (gender, age, sexual orientation, region of origin, personality profile) and the document (timestamp, genre, topic, veracity, sentiment). 80% of the reviews are written by females, 66% by authors from Antwerpen and the rest from other parts of Belgium and the Netherlands.

On the FourSquare and CSI datasets, the model, which is based on [114], first uses an embedding layer to turn non-negative integers (locations indices and word tokens) into dense vectors of dimension 320, then applies three spatial convolutional layers with 100 filters and variable kernel windows of size $(3, 320)$, $(4, 320)$ and $(5, 320)$ and max pooling layers with pooling size set to $(l-3, 1)$, $(l-4, 1)$, and $(l-5, 1)$ where $l$ is the fixed length to which input sequences are padded. The hyper-parameter $l$ is 300 on CSI and 100 on FourSquare. After this, the model has two fully connected layers of size 128 and 2 for FourSquare

| Yelp-health | | FourSquare | |
|---|---|---|---|
| **Batch Size** | **Precision** | **Batch Size** | **Precision** |
| 32 | 0.92 | 100 | 0.99 |
| 64 | 0.84 | 200 | 0.98 |
| 128 | 0.75 | 500 | 0.91 |
| 256 | 0.66 | 1,000 | 0.76 |
| 512 | 0.62 | 2,000 | 0.62 |

**Table 7.2:** Precision of membership inference (recall is 1).

and one fully connected layer of size 2 for CSI. We use RELU as the activation function. Batch size is 100 for FourSquare, 12 for CSI. SGD learning rate is 0.01.

**Experimental setup.** Our experiments have been performed on a workstation running Ubuntu Server 16.04 LTS equipped with a 3.4GHz CPU i7-6800K, 32GB RAM, and an NVIDIA TitanX GPU card. We use MxNet [50] and Lasagne [64] to implement deep neural networks and Scikit-learn [169] for conventional machine learning models. Training our inference models takes less than 60 seconds on average and does not require a GPU.

**Metrics.** We use AUC scores to evaluate the performance of the collaborative model and of our property inference. For membership inference, we report only precision because our decision rule from Section 7.1.3 is binary and does not produce a probability score.

### 7.2.2 Two-party membership inference

The adversary first builds a Bag of Words (BoW) representation for the input whose membership in the target's training data he aims to infer. We denote this as the test BoW. During training, as explained in Section 7.1.3, the non-zero gradients of the embedding layer reveal which "words" are present in each batch of the target's data, enabling the adversary to build a BoW. If the test BoW is a subset of the batch BoW, the adversary concludes that the

input in question is included in this batch.

To demonstrate membership inference, we choose Yelp-health and FourSquare datasets and set the vocabulary to the 5,000 most frequent words and 30,000 most popular locations, respectively. We split the data evenly between the target and the adversary and train a collaborative model for 3,000 iterations.

Table 7.2 shows the precision of membership inference for different batch sizes. As batch sizes increase, the adversary observes more words in each batch BoW and the attack produces more false positives. Recall is always perfect, i.e., there are no false negatives, because any true test BoW must be contained in at least one of the batch BoWs observed by the adversary.

### 7.2.3 Two-party single-batch property inference

Next, we infer properties of the target participant's training batches. We call a batch $B_{\text{nonprop}}$ if none of the inputs in it have the property, $B_{\text{prop}}$ otherwise. The adversary's goal is to identify, by observing gradient updates, which of the batches are $B_{\text{prop}}$.

We split the training data evenly between the target and the adversary and assume that same fraction in both subsets has the property. During training, $\frac{1}{m}$ of the target's batches include only inputs with the property ($m = 2$ in the following experiments).

**LFW.** Table 7.3 reports the results of single-batch property inference on the LFW dataset. For the inference tasks, we choose properties that are *uncorrelated* with the main classification label that the collaborative model is trying to learn. The attack has perfect AUC when the main task is gender classification and the inference task is "race:black" (these labels are independent; their Pearson correlation is -0.005). The attack also achieves almost perfect AUC when the main task is "race: black" and the inference task is "eyewear: sunglasses". It also performs well on several other properties, including "eyewear: glasses" when the main task is "race: Asian".

These results demonstrate that gradients observed during training leak

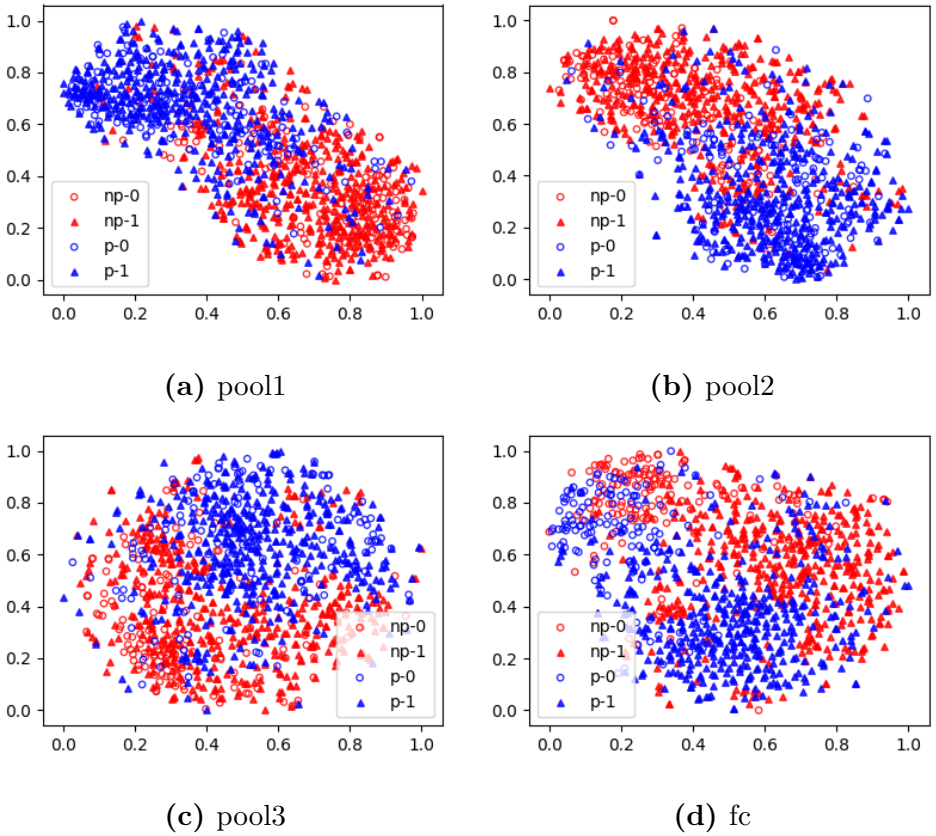
| Main T. | Infer T. | Corr. | AUC | Main T. | Infer T. | Corr. | AUC |
|---|---|---|---|---|---|---|---|
| Gender | Black | -0.005 | 1.0 | Gender | Sunglasses | -0.025 | 1.0 |
| Gender | Asian | -0.018 | 0.93 | Gender | Eyeglasses | 0.157 | 0.94 |
| Smile | Black | 0.062 | 1.0 | Smile | Sunglasses | -0.016 | 1.0 |
| Smile | Asian | 0.047 | 0.93 | Smile | Eyeglasses | -0.083 | 0.97 |
| Age | Black | -0.084 | 1.0 | Race | Sunglasses | 0.026 | 1.0 |
| Age | Asian | -0.078 | 0.97 | Race | Eyeglasses | -0.116 | 0.96 |
| Eyewear | Black | 0.034 | 1.0 | Hair | Sunglasses | -0.013 | 1.0 |
| Eyewear | Asian | -0.119 | 0.91 | Hair | Eyeglasses | 0.139 | 0.96 |

**Table 7.3:** AUC score of single-batch property inference on LFW. We also report the Pearson correlation between the main task label and the property label.

more than the characteristic features of each class. In fact, collaborative learning leaks properties of the training data that are uncorrelated with class membership. To understand why, we plot the t-SNE projection [136] of the features from different layers of the joint model in Figure 7.3. Observe that the feature vectors are grouped by property in the lower layers pool1, pool2 and pool3, and by class label in the higher layer. Intuitively, this shows that the model does not just learn to separate inputs by class. The lower layers of the model learn to separate inputs by various properties that are irrelevant for the model's designated task. Our inference attack exploits this unintended extra functionality, which also shows the effective model capacity of neural networks to memorize the dataset [223].

**Yelp-health.** On this dataset, we use review score classification (specifically, sentiment analysis) as the main task and the specialty of the doctor being reviewed as the property inference task. Obviously, the latter is more sensitive from the privacy perspective.

We use 3,000 most frequent words in the corpus as the vocabulary and train the model for 3,000 iterations. Using BoWs from the embedding-layer gradients, the attack achieves almost perfect AUC for inferring the doctor




**(a)** pool1 **(b)** pool2

**(c)** pool3 **(d)** fc

**Figure 7.3:** t-SNE projection of the features from different layers of the joint model on LFW gender classification; 0 is female, 1 is male. The property (i.e., the blue points denoted by p-0 and p-1) is "race: black", while the red points without the property are denoted by np-0 and np-1

specialty. Table 7.4 shows the words that have the highest predictive power in our logistic regression.

**Fractional properties.** We also investigate if it is possible to infer the property when only some of the inputs in a batch have it. For these experiments, we use FaceScrub's top 5 face IDs and Yelp-author (the latter with the 3,000 most frequent words as the vocabulary). The model is trained for 3,000 iterations. As before, $1/m$ of the target's batches include inputs with the property ($m = 2$ in the experiments), but here we vary the fraction of the inputs in the batch that have the property among 0.1, 0.3, 0.5, 0.7, and 0.9.

Figure 7.4 reports the results. On FaceScrub for IDs 0, 1, and 3, AUC

| Health Service | Top Words in Positive Class |
|---|---|
| Obstetricians | pregnancy, delivery, women, birth, ultrasound |
| Pediatricians | pediatrics, sick, parents, kid, newborn |
| Cosmetic Surgeons | augmentation, plastic, breast, facial, implants |
| Cardiologists | cardiologist, monitor, bed, heart, ER |
| Dermatologists | acne, dermatologists, mole, cancer, spots |
| Ophthalmologists | vision, LASIK, contacts, lenses, frames |
| Orthopedists | knee, orthopedic, shoulder, injury, therapy |
| Radiologists | imaging, SimonMed, mammogram, CT, MRI |
| Psychiatrists | psychiatrist, mental, Zedek, depression, sessions |
| Urologists | Edgepark, pump, supplies, urologist, kidney |

**Table 7.4:** Words with the largest positive coefficients in the property classifier for Yelp-health.

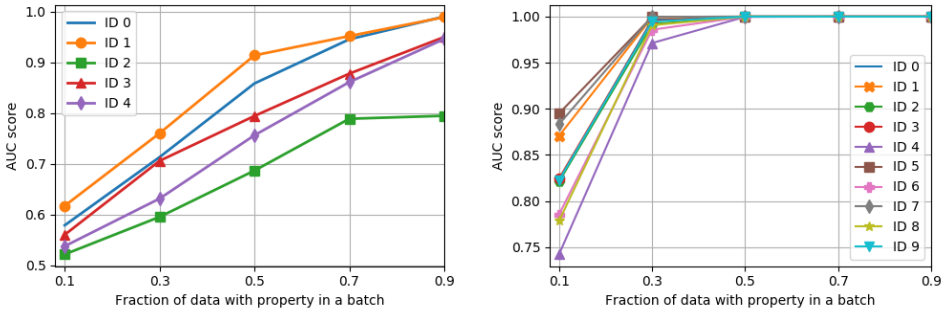
scores are above 0.8 even if only 50% of the batch contain that face. This means that the adversary can successfully infer that photos of a particular person appear in a batch even though (a) the model is trained for generic gender classification, and (b) half of the photos in the batch are of other people. If the fraction is higher, AUC scores approach 1.

On Yelp-author, AUC scores are above 0.95 for all identities even when the fraction is 0.3. This means that we can successfully identify the authors of reviews even though (a) the model is trained for generic sentiment analysis, and (b) more than 2/3rd of the reviews in the batch are from other authors.

Both results concretely illustrate that collaborative learning leaks much more than the characteristic features of each class.

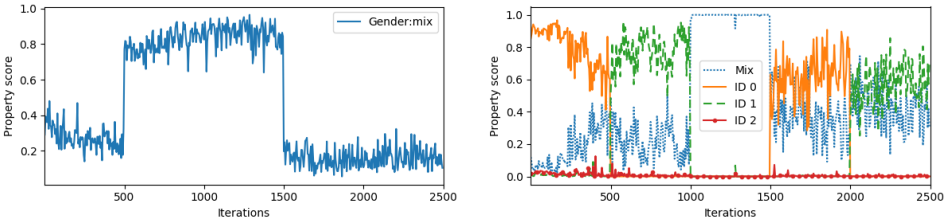
### 7.2.4 Inferring when a property occurs

Continuous training, when new training data is added to the process as it becomes available, presents interesting opportunities for inference attacks. If the occurrences of a property in the training data can be linked to events




**(a)** FaceScrub **(b)** Yelp-author

**Figure 7.4:** AUC vs. the fraction of the batch that has the property on FaceScrub and Yelp-author.




**(a)** PIPA **(b)** FaceScrub

**Figure 7.5:** Detecting occurrence of a single-batch property.

outside the training process, privacy leakage is exacerbated. For example, if the adversary can infer that a certain third person started appearing in another participant's training data immediately after that participant uploaded his photos from a trip, this may constitute a serious privacy breach.

**PIPA.** Images in the PIPA dataset have between 1 to 3 faces. We train the collaborative model to detect if there is a young adult in the image; the adversary's inference task is to determine if the people in the image are of the same gender. The latter property is a stepping stone to inferring social relationships, and thus is sensitive. We train the model for 2,500 iterations and let the batches with the "same gender" property appear in iterations 500 to 1500.

Figure 7.5(a) shows, for each iteration, the probability output by the adversary's classifier that the batch in that iteration has the property. The

appearance and disappearance of the property in the training data are clearly visible in the plot.

**FaceScrub.** For the gender classification model on FaceScrub, the adversary's objective is to infer whether and when a certain person appears in a participant's photos.

The joint model is trained for 2,500 iterations. We arrange the target's training data so that two specific identities appear during certain iterations: ID 0 in iterations 0 to 500 and 1500 to 2000, ID 1 in iterations 500 to 1000 and 2000 to 2500. In the other iterations, the batches are mixtures of other identities. The adversary trains three property classifiers, for ID 0, ID 1, and a third ID that does not appear in the target's dataset (ID 2).

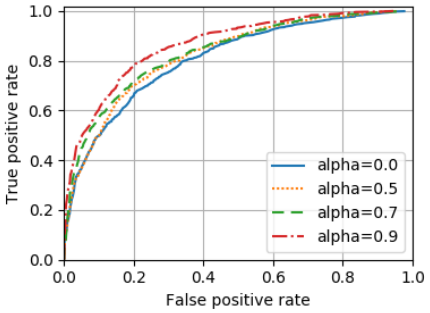
Figure 7.5(b) reports the scores of all three classifiers. ID 0 and 1 receive the highest scores in the iterations where they appear, whereas ID 2, which never appears in the training data, receives very low scores in all iterations.

These experiments show that our attacks can successfully infer *dynamic properties* of the training dataset as collaborative learning progresses.

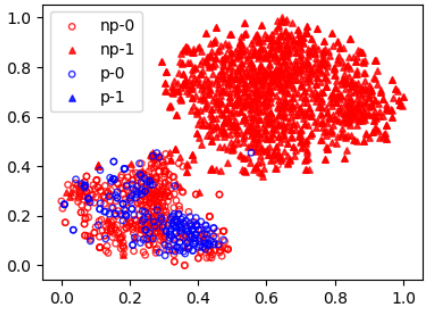
### 7.2.5 Active property inference

To show the additional power of the active attack from Section 7.1.5, we use FaceScrub. The main task is gender classification, the adversary's task is to infer the presence of ID 4 in the training data. We assume that this ID occurs in a single batch, where it constitutes 50% of the photos. We evaluate the attack with different choices of $\alpha$, which controls the balance between the main-task loss and the property-classification loss in the adversary's objective function.

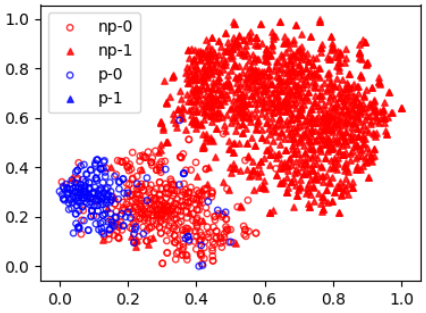
Figure 7.6(a) shows that AUC increases as we increase $\alpha$. Figure 7.6(b) and Figure 7.6(c) show the t-SNE projection of the final fully connected layer, with $\alpha = 0$ and $\alpha = 0.7$, respectively. Observe that the data with the property (blue points) is grouped tighter when $\alpha = 0.7$ than in the model trained under a passive attack ($\alpha = 0$). This illustrates that as a result of the active attack, the joint model learns a better separation for data with and without the property.




**(a)** ROC for different $\alpha$




**(b)** t-SNE of the final layer for $\alpha = 0$



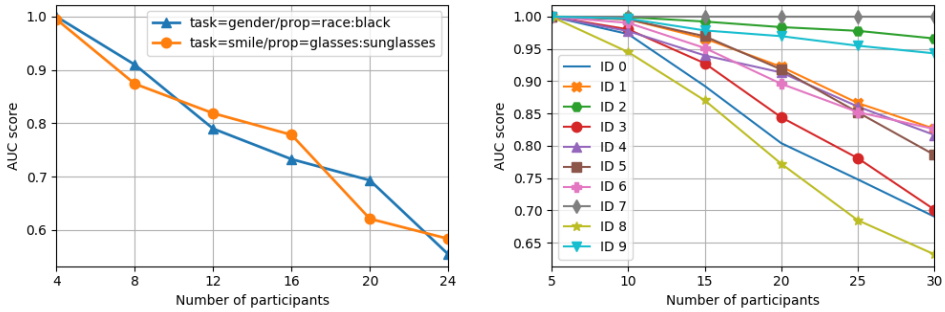

**(c)** t-SNE of the final layer for $\alpha = 0.7$

**Figure 7.6:** Active property inference attack on FaceScrub.

### 7.2.6 Multi-party with synchronized SGD

As the number of honest participants in collaborative learning (cf. Algorithm 1) increases, the adversary's task becomes harder because the observed gradient updates are aggregated across multiple participants. Furthermore, the inferred information may not directly reveal the identity of the participant to whom the data belongs, although auxiliary information can help in this case (see Section 7.1.1).

We evaluate our attacks on LFW and Yelp-author datasets. We split the training data evenly across all participants, but in such a way that only the target and the adversary have the data with the property. The joint model is trained with the same hyper-parameters as in the two-party case. Similar to the two-party setting (see Section 7.2.3), the adversary's goal is to identify which aggregated gradient updates are based on batches $B_{\text{prop}}$ with the property.




**(a)** LFW **(b)** Yelp-author

**Figure 7.7:** Multi-party with synchronized SGD: attack AUC score vs. the number of participants.

**LFW.** We experiment with (1) gender classification as the main task and "race: black" as the inference task, and (2) smile classification as the main task and "eyewear: sunglasses" as the inference task.

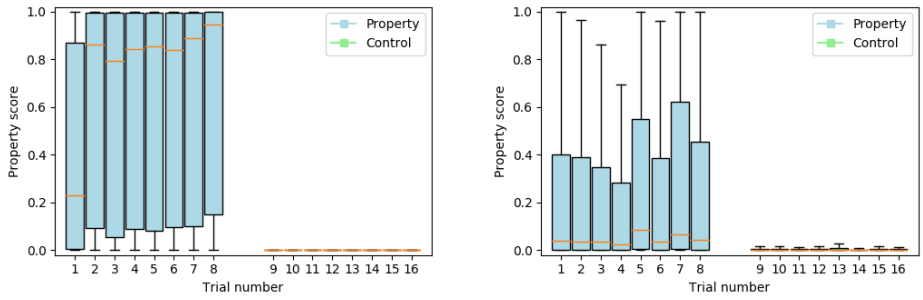
Figure 7.7(a) shows that the attack still achieves reasonably high performance, with AUC score around 0.8, when the number of participants is 12. Performance then degrades for both tasks.

**Yelp-author.** The inference task is again author identification. In the multi-party case, the gradients of the embedding layer leak the batch BoWs of all honest participants, not just the target.

Figure 7.7(b) reports the results. For some authors, the AUC scores do not degrade significantly even with many participants. This is likely due to some unique combinations of words used by these authors, which identify them even in multi-party settings.

### 7.2.7 Multi-party with model averaging

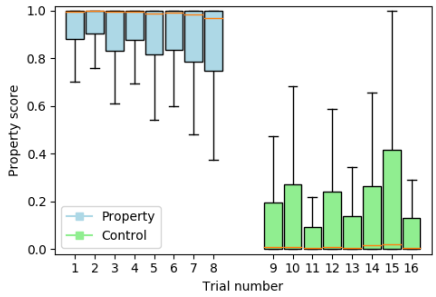
In every round $t$ of multi-party federated learning with model averaging, the adversary observes $\theta_t - \theta_{t-1} = \sum_k \frac{n^k}{n}\theta_t^k - \sum_k \frac{n^k}{n}\theta_{t-1}^k = \sum_k \frac{n^k}{n}(\theta_t^k - \theta_{t-1}^k)$, where $\theta_t^k - \theta_{t-1}^k$ are the aggregated gradients computed on the $k$-th participant's local dataset. To simplify our experiments and because we do not care about the computational efficiency of the learning protocol for the purposes of



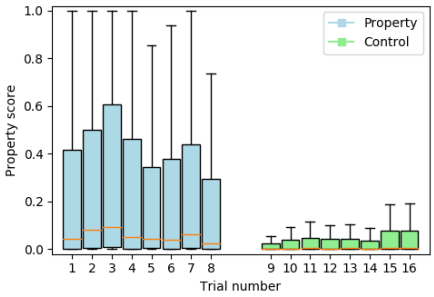

**(a)** Face ID 1, $K = 3$




**(b)** Face ID 1, $K = 5$




**(c)** Face ID 3, $K = 3$




**(d)** Face ID 3, $K = 5$

**Figure 7.8:** Multi-party collaborative training with model averaging: box plots show the distribution of the adversary's scores in each trial. In the 8 trials on the left, one of the participants' data has the property; in the 8 trials on the right, none of the honest participants have the data with the property.

our analysis, we set $C = 1$, i.e., the server takes updates from all clients in all rounds (see Algorithm 2 in Section 2.3.5).

In our experiments, we split the training data evenly among honest participants but ensure that in the target participant's subset, $\hat{p}\%$ of the inputs have the property while none of the other honest participants' data has the property. During each epoch of local training, every honest participant splits his local training data into 10 batches and performs one round of training.

We assume that the adversary has the same number of inputs with the property as the target. As before, when the adversary trains his binary classifier, he needs to locally "emulate" the collaborative training process, i.e., sample data from his local dataset, compute aggregated updates, and learn to distinguish between the aggregates based on the data without the property

and aggregates where one of the underlying updates was based on the data with the property.

We perform this experiment for 8 trials. For the control experiments, we set $\hat{p} = 0\%$ and also perform them 8 times. We want to see if it is possible to distinguish the trials where there is a subset in the training data that has the property from the control trials where there are no training inputs with the property.

**Detecting presence of a face in the training data.** We use FaceScrub and select two face IDs (1 and 3) whose presence we want to infer. In the "property" case, $\hat{p} = 80\%$, i.e., 80% of one honest participant's training data consist of the photos that depict the person in question. In the "control" case, $\hat{p} = 0\%$, i.e., the person does not occur in the training data of any honest participants.

Figure 7.8 shows the scores assigned by the adversary's classifier to the aggregated updates with 3 and 5 total participants. When the face of interest is present in the training dataset, the scores are much higher than when it is absent.

Inference is a probabilistic process and there are no guarantees. Success of the attack depends on the property being inferred, distribution of the data across participants, and other factors. For example, the classifiers for Face IDs 2 and 4, which were trained in the same fashion as the classifiers for Face IDs 1 and 3, failed to infer the presence of "their" faces in the training data.

**Inferring when a property occurs.** In this experiment, we aim to infer when a participant whose local data has a certain property joined collaborative training. We first let the adversary and the rest of the honest participants train the joint model for 250 rounds. The target participant then joins the training at round $t = 250$ with the local data that consists of photos depicting ID 1.

Figure 7.9 shows the results. The adversary's AUC scores are around 0 when images with face ID 1 are not present and then increase almost to 1.0 right after the target participant joins the training.

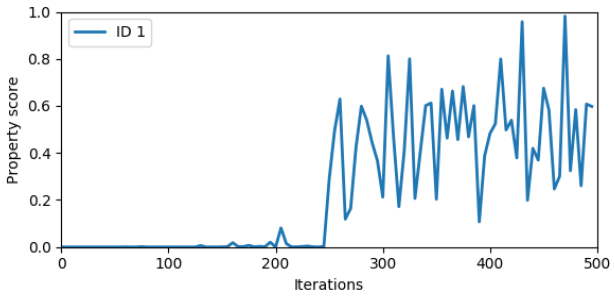



**Figure 7.9:** Detecting when a participant whose local data has the property of interest joins the training. $K = 2$ for rounds 0 to 250, $K = 3$ for rounds 250 to 500.

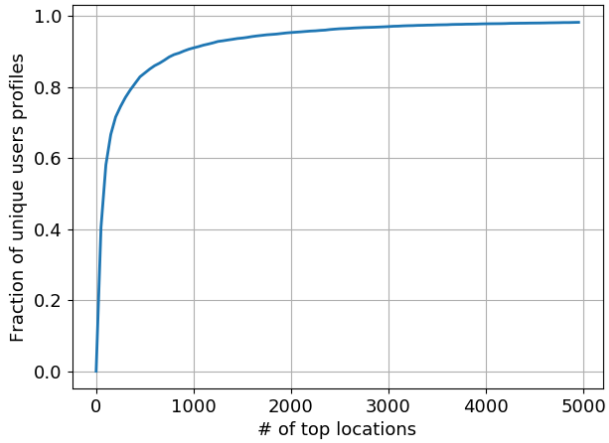



**Figure 7.10:** Uniqueness of user profiles with respect to the number of top locations.

## 7.3 Defenses

We now discuss and evaluate possible defenses.

### 7.3.1 Selective gradient sharing

As suggested in [190], participants in collaborative learning could share only a *fraction* of their gradients during each update. This reduces communication overhead and, potentially, leakage, since the adversary observes fewer gradients.

To evaluate this defense, we measure the performance of single-batch inference against a sentiment classifier collaboratively trained on the CSI Corpus by two parties who exchange only a fraction of their gradients. Table 7.5 shows

| Property / % parameters update | 10% | 50% | 100% |
|---|---|---|---|
| Top region (Antwerpen) | 0.84 | 0.86 | 0.93 |
| Gender | 0.90 | 0.91 | 0.93 |
| Veracity | 0.94 | 0.99 | 0.99 |

**Table 7.5:** Inference attacks against the CSI Corpus for different fractions of gradients shared during training.

the resulting AUC scores. When inferring the region of the texts' authors, our attack still achieves 0.84 AUC when only 10% of the updates are shared during each iteration, compared to 0.93 AUC when all updates are shared.

### 7.3.2 Dimensionality reduction

As discussed in Section 7.1.2, if the input space of the model is very sparse and the inputs must be embedded into a lower-dimensional space, non-zero gradient updates in the embedding layer reveal which input tokens are present in the training batch.

One plausible defense is to only use inputs that occur many times in the training data. This does not work in general: for example, Figure 7.10 shows that restricting inputs to the top locations in the FourSquare dataset eliminates most of the training data.

A smarter defense is to restrict the model so that it only uses "words" from a pre-defined vocabulary of common words. For example, Google's federated learning for predictive keyboards uses a fixed vocabulary of 5,000 words [143].

In Table 7.6, we report the accuracy of our membership inference attack and of the joint model on its main task—gender classification for the FourSquare dataset, sentiment analysis for the CSI Corpus—for different sizes of the common vocabulary (locations in the case of FourSquare, words in the case of CSI). Overall, this approach partially mitigates our attacks but also has a significant negative impact on the quality of the collaboratively trained models.

| CSI | | | FourSquare | | |
|---|---|---|---|---|---|
| **Top N words** | **Attack Precision** | **Model AUC** | **Top N locations** | **Attack Precision** | **Model AUC** |
| 4,000 | 0.94 | 0.91 | 30,000 | 0.91 | 0.64 |
| 2,000 | 0.92 | 0.87 | 10,000 | 0.86 | 0.59 |
| 1,000 | 0.92 | 0.85 | 3,000 | 0.65 | 0.51 |
| 500 | 0.82 | 0.84 | 1,000 | 0.52 | 0.50 |

**Table 7.6:** Membership inference against the CSI Corpus and FourSquare for different vocabulary sizes.

### 7.3.3 Dropout

Another possible defense is to employ *dropout* [194], a popular technique often used as a regularizer to mitigate overfitting in neural networks. Dropout randomly deactivates activations between neurons, with probability $p_{drop} \in [0, 1]$. Random deactivations may weaken our attacks because the adversary observes fewer gradients corresponding to the active neurons.

To evaluate this approach, we add dropout after the max pool layers in the joint model. Table 7.7 reports the accuracy of inferring the region of the reviews in the CSI Corpus, for different values of $p_{drop}$. Increasing the randomness of dropout actually makes our attacks stronger while slightly decreasing the accuracy of the joint model. We believe this is due to the increased variance between participants' updates, which yields more robust features for the adversary's inference model.

### 7.3.4 Participant-level differential privacy

Record-level $\varepsilon$-differential privacy, by definition, bounds the success of membership inference but does not prevent property inference. Any application of differential privacy entails application-specific tradeoffs between privacy of the training data and accuracy of the resulting model. The participants must also somehow choose the parameters (e.g., $\varepsilon$) that control this tradeoff.

| Dropout Prob. | Attack AUC | Model AUC |
|---|---|---|
| 0.1 | 0.94 | 0.87 |
| 0.3 | 0.97 | 0.87 |
| 0.5 | 0.98 | 0.87 |
| 0.7 | 0.99 | 0.86 |
| 0.9 | 0.99 | 0.84 |

**Table 7.7:** Inference of the top region (Antwerpen) on the CSI Corpus for different values of dropout probability.

In theory, participant-level differential privacy bounds the success of all inference attacks described in this work. We implemented the participant-level differentially private federated learning algorithm by McMahan et al. [144] and attempted to train a gender classifier on LFW. However, the model did not converge for any number of participants (we tried at most 30). This is due to the magnitude of noise needed to achieve differential privacy with the moments accountant bound [1], which is inversely proportional to the number of users (the model in [144] was trained on *thousands* of users). Another participant-level differential privacy mechanism, presented in [85], also requires a very large number of participants.

Therefore, it remains an open research question whether or not it is possible to (1) prevent our inference attacks *and* (2) obtain high-quality models by training with participant-level differential privacy and relatively few (e.g., dozens) participants.

### 7.3.5 Sensitivity to number of training epochs

Finally, we measure the sensitivity of our attacks to the number of training epochs. Figure 7.11 reports the accuracy of inferring the author's gender in the CSI Corpus vs. the number of full training epochs (in a two-party setting).

We find that our attack is very effective, reaching 0.98 AUC after only 2 epochs and improving as the training progresses and the adversary collects

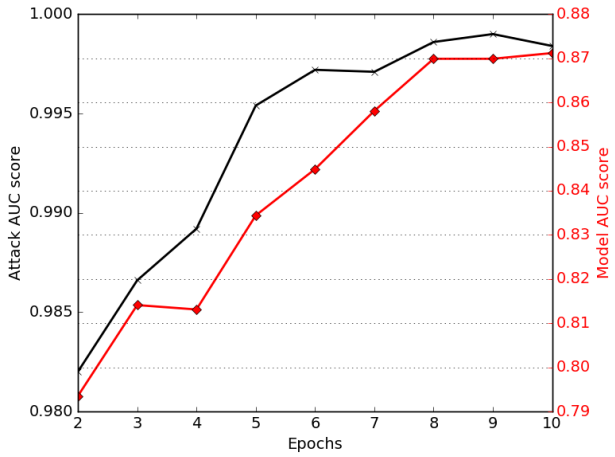



**Figure 7.11:** Attack performance with respect to the number of collaborative learning epochs.

more gradient updates, while the performance of the main model is not affected.

## Chapter 8

# Conclusion

Large-scale data collection efforts open the door to possible privacy breaches, while fears of privacy harm often prevent the development of useful machine learning applications. To address the tension between the utility of extracting knowledge from data and the duty to protect individuals' sensitive information, this dissertation presented several results in the design, development, and evaluation of privacy-aware data processing systems.

First, we introduced efficient techniques for privately and efficiently collecting statistics (Chapter 4). By relying on private data aggregation protocols and succinct data structures, we showed how to reduce the communication and computation complexity incurred by each data source from linear to logarithmic in the size of the input, while only introducing a limited, upper-bounded error in the quality of the statistics. Our techniques support different trust, robustness, and deployment models and can be applied to a number of interesting real-world problems where aggregate statistics are used to train machine learning models.

Then, we presented a novel differentially private generative model, relying on a mixture of $k$ generative neural networks (Chapter 5). The training data is first partitioned into $k$ parts using a differentially private kernel $k$-means, then each cluster is given to a separate generative neural network, such as RBM or VAE, which are trained only on their own cluster using differentially private gradient descent. The trained models can be used to generate and share

synthetic high-dimensional data with provable privacy. We have evaluated the performance of the model on real-world datasets, showing that our approach provides accurate representation of large datasets with strong privacy guarantees and high utility.

Next, we proposed a first-of-its-kind evaluation of information leakage in generative models, showing that a variety of models are vulnerable to *membership* inference attacks, i.e., the presence of exact data points in training data (Chapter 6). We showed that our attacks can be used to detect overfitting in generative models and help selecting an appropriate model that will not leak information about samples on which it was trained. Moreover, we demonstrated that we can infer membership using a novel method for training GANs, and that an attacker with limited auxiliary knowledge of dataset samples can remarkably improve their accuracy.

Finally, we proposed and evaluated several inference attacks against collaborative deep learning (Chapter 7). These attacks enable a malicious participant to not only infer membership, but also *properties* that characterize subsets of the training data and are independent of the properties that the joint model aims to capture. We found that the root cause of these attacks is that when a deep learning model is trained, it internally learns to recognize many more features of the data than is necessary for the task that it is being trained for. Consequently, model updates during collaborative learning leak information about these extra, "unintended" features to adversarial participants. Active attacks are potentially very powerful in this setting because they enable the adversary to trick the joint model into learning features of the adversary's choosing without a significant impact on the model's performance on its main task.

Progress in machine learning has been enabled by the ability of analyzing massive amounts of data and refining model parameters to better encode the patterns within that data. On the one hand, machine learning models should learn general patterns of the training data in order to be able to generalize

with new unseen data. On the other hand, properties of single training data-points or subsets of data-points belonging to specific individuals should not be memorized and then revealed to avoid violating users' privacy. Unfortunately, prior work [189], as well as novel contributions presented in this dissertation, show that machine learning models lead to information leakage about users' training data. However, strong connections between privacy breaches and overfitting lead to the conclusion that privacy mechanisms such as differential privacy can work together with training regularization techniques to achieve common objectives between the fields of Privacy Enhancing Technologies and Machine Learning.

We conclude this dissertation by highlighting some open problems and items for future work.

**Privately training machine learning models.** In Chapter 4, we showed how to privately and efficiently collect data from large streams, and then use the aggregate to extract useful estimates of statistics and train simple machine learning models. One interesting research direction is to investigate scenarios in which different succinct data structures could lead to better accuracy-efficiency tradeoffs, thus allowing the training of more sophisticated statistical models, e.g., learning word embeddings [150].

In Chapter 5, we introduced a two-step training process for clustering generative neural network models. However, clustering algorithms are often dependent on the type of the input data, thus limiting the performance of our proposal to different domains. Therefore, the design of general frameworks that jointly optimize the generative network model and clustering assignments [10], albeit with strong privacy guarantees, remains a challenging topic for further research. Also, the effective privacy-preserving training of more advanced deep neural networks is also desirable in order to generate more complex data such as high-resolution photos [112].

Finally, it would be interesting to deploy our techniques in the wild, with the release of open-source frameworks supporting large-scale privacy-

preserving aggregation and generative machine learning as a service.

**Privacy leakage in machine learning.** Results presented in Chapters 6 and 7 suggest that inference attacks are a realistic threat in both generative neural network models and collaborative learning. Nonetheless, we identify some limitations that motivate the need for further research. For instance, our attacks against generative models assume the attacker knows the size of the training set, which limits their application to only specific scenarios. Relaxing this assumption remains an interesting topic to be explored, along with evaluating inference attacks on (differentially private) generative models which are not based on GANs. Moreover, our membership inference attacks against collaborative learning can only be applied to embedding layers, thus motivating the need for evaluating different layers in neural networks.

We also showed that defenses such as selective gradient sharing, reducing dimensionality, and dropout are not always effective. This should motivate future work on better defenses. For instance, techniques that learn only the features relevant to a given task [76, 163] can potentially serve as the basis for "least-privilege" collaboratively trained models. Further, methods could be developed to spot active attacks that manipulate the model into learning extra features. Finally, it remains an open question if participant-level differential privacy mechanisms can produce accurate models when collaborative learning involves relatively few participants.

# Bibliography


[1] Martín Abadi et al. “Deep learning with differential privacy”. In: *ACM CCS*. 2016.

[2] John M Abowd et al. “How protective are synthetic data?” In: *PSD*. 2008.

[3] Gergely Acs et al. “Privacy-Preserving Data Release with Generative Neural Networks”. In: *IEEE ICDM*. 2017.

[4] Gergely Ács et al. “I have a dream!(differentially private smart metering)”. In: *ACM IH*. 2011.

[5] Gediminas Adomavicius et al. “Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions”. In: *IEEE TKDE* (2005).

[6] Charu C Aggarwal. “On k-anonymity and the curse of dimensionality”. In: *VLDB*. 2005.

[7] Istemi Ekin Akkus et al. “Non-tracking Web Analytics”. In: *ACM CCS*. 2012.

[8] Mohammad Alaggan et al. “BLIP: non-interactive differentially-private similarity computation on bloom filters”. In: *SSS*. 2012.

[9] Mohammad Alaggan et al. “Privacy-preserving Wi-Fi Analytics”. In: *PETS* (2018).

[10] Elie Aljalbout et al. “Clustering with Deep Learning: Taxonomy and New Methods”. In: *arXiv preprint 1801.07648* (2018).

[11] Yoshinori Aono et al. “Privacy-preserving deep learning: Revisited and Enhanced”. In: *ATIS*. 2017.

[12] Martin Arjovsky et al. “Wasserstein gan”. In: *arXiv preprint 1701.07875* (2017).

[13] Hassan Jameel Asghar et al. “SplitBox: Toward Efficient Private Network Function Virtualization”. In: *ACM HotMiddlebox*. 2016.

[14] Vikas G Ashok et al. “A Scalable and Efficient Privacy Preserving Global Itemset Support Approximation Using Bloom Filters”. In: *DBSEC*. 2014.

[15] Giuseppe Ateniese et al. “Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers”. In: *IJSN* (2015).

[16] Haim Avron et al. “Random Fourier features for kernel ridge regression: Approximation bounds and statistical guarantees”. In: *arXiv preprint 1804.09893* (2018).

[17] Michael Backes et al. “Membership Privacy in MicroRNA-based Studies”. In: *ACM CCS*. 2016.

[18] Raef Bassily et al. “Local, Private, Efficient Protocols for Succinct Histograms”. In: *STOC*. 2015.

[19] Raef Bassily et al. “Private empirical risk minimization: Efficient algorithms and tight error bounds”. In: *FOCS*. 2014.

[20] BBC. *Google DeepMind NHS app test broke UK privacy law*. `http://www.bbc.com/news/technology-40483202`. 2017.

[21] Brett K Beaulieu-Jones et al. “Privacy-preserving generative deep neural networks support clinical data sharing”. In: *bioRxiv* (2017).

[22] Josh Benaloh. “Dense probabilistic encryption”. In: *SAC*. 1994.

[23] Yoshua Bengio et al. “Generalized denoising auto-encoders as generative models”. In: *NIPS*. 2013.

[24] Daniel J Bernstein et al. “High-speed High-Security Signatures”. In: *CHES*. 2011.

[25] David Berthelot et al. “BEGAN: Boundary Equilibrium Generative Adversarial Networks”. In: *arXiv preprint 1703.10717* (2017).

[26] Giuseppe Bianchi et al. “”Better Than Nothing” Privacy with Bloom Filters: To What Extent?” In: *PSD*. 2012.

[27] Vincent Bindschaedler et al. “Plausible Deniability for Privacy-Preserving Data Synthesis”. In: *VLDB* (2017).

[28] Burton H Bloom. “Space/time trade-offs in hash coding with allowable errors”. In: *Communications of the ACM* 13.7 (1970).

[29] Avrim Blum et al. “Practical privacy: the SuLQ framework”. In: *PODS*. 2005.

[30] Keith Bonawitz et al. “Practical Secure Aggregation for Privacy-Preserving Machine Learning”. In: *ACM CCS*. 2017.

[31] Joppe W Bos et al. “Private predictive analysis on encrypted medical data”. In: *Journal of Biomedical Informatics* (2014).

[32] Raphael Bost et al. *Machine learning classification over encrypted data*. Tech. rep. Cryptology ePrint Archive Report 2014/331, 2014.

[33] Claire McKay Bowen et al. “Comparative Study of Differentially Private Data Synthesis Methods”. In: *arXiv preprint 1602.01063* (2016).

[34] Justin Brickell et al. “The cost of privacy: destruction of data-mining utility in anonymized data publishing”. In: *KDD*. 2008.

[35] Martin Burkhart et al. “SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics”. In: *Usenix Security*. 2010.

[36] Joseph A Calandrino et al. ““You Might Also Like:” Privacy Risks of Collaborative Filtering”. In: *IEEE S&P*. 2011.

[37] Nicholas Carlini et al. “The Secret Sharer: Measuring Unintended Neural Network Memorization & Extracting Secrets”. In: *arXiv preprint 1802.08232* (2018).

[38] David E Carlson et al. “Stochastic Spectral Descent for Restricted Boltzmann Machines.” In: *AISTATS*. 2015.

[39] Claude Castelluccia et al. “Efficient Aggregation of encrypted data in Wireless Sensor Networks”. In: *Mobiquitous*. 2005.

[40] Pablo Samuel Castro et al. “Urban traffic modelling and prediction using large scale taxi GPS traces”. In: *IEEE PerCom*. 2012.

[41] T-H Hubert Chan et al. “Differentially private continual monitoring of heavy hitters from distributed streams”. In: *PETS*. 2012.

[42] T-H Hubert Chan et al. “Privacy-preserving stream aggregation with fault tolerance”. In: *FC*. 2012.

[43] Anne-Sophie Charest. “How can we analyze differentially-private synthetic datasets?” In: *JPC* (2011).

[44] Moses Charikar et al. “Finding frequent items in data streams”. In: *ICALP*. 2002.

[45] Amir Chaudhry et al. “Personal data: thinking inside the box”. In: *Fifth Decennial Aarhus Conference on Critical Alternatives*. 2015.

[46] Kamalika Chaudhuri et al. “Differentially private empirical risk minimization”. In: *JMLR* (2011).

[47] Ruichuan Chen et al. “SplitX: High-performance Private Analytics”. In: *SIGCOMM*. 2013.

[48] Ruichuan Chen et al. “Towards statistical queries over distributed private user data”. In: *NSDI*. 2012.

[49] Rui Chen et al. “Publishing set-valued data via differential privacy”. In: *VLDB* (2011).

[50] Tianqi Chen et al. "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems". In: *arXiv preprint 1512.01274* (2015).

[51] Trishul M Chilimbi et al. "Project Adam: Building an Efficient and Scalable Deep Learning Training System." In: *OSDI*. 2014.

[52] Soumith Chintala et al. *How to Train a GAN? Tips and tricks to make GANs work*. `https://github.com/soumith/ganhacks`.

[53] Radha Chitta et al. "Efficient Kernel Clustering Using Random Fourier Features". In: *ICDM*. 2012.

[54] Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *EMNLP*. 2014.

[55] Thomas H Cormen et al. *Introduction to algorithms*. MIT Press Cambridge, 2001.

[56] Graham Cormode et al. "An Improved Data Stream Summary: The Count-Min Sketch and Its Applications". In: *Journal of Algorithms* (2005).

[57] Graham Cormode et al. "Differentially private summaries for sparse data". In: *ICDT*. 2012.

[58] Henry Corrigan-Gibbs et al. "Dissent: accountable anonymous group messaging". In: *ACM CCS*. 2010.

[59] Henry Corrigan-Gibbs et al. "Prio: Private, Robust, and Scalable Computation of Aggregate Statistics." In: *NSDI*. 2017.

[60] Henry Corrigan-Gibbs et al. "Proactively Accountable Anonymous Messaging in Verdict." In: *USENIX Security*. 2013.

[61] Henry Corrigan-Gibbs et al. "Riposte: An anonymous messaging system handling millions of users". In: *IEEE S&P*. 2015.

[62] *Count-Min Sketch and its applications*. `https://sites.google.com/site/countminsketch/`. 2015.

[63] Jeffrey Dean et al. “Large scale distributed deep networks”. In: *NIPS*. 2012.

[64] Sander Dieleman et al. *Lasagne: First Release*. `http://dx.doi.org/10.5281/zenodo.27878`. 2015.

[65] Roger Dingledine et al. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[66] Kamran Ghasedi Dizaji et al. “Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization”. In: *ICCV*. 2017.

[67] Nathan Dowlin et al. “Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy”. In: *ICML*. 2016.

[68] Wenliang Du et al. “Privacy-preserving multivariate statistical analysis: Linear regression and classification”. In: *IEEE ICDM*. 2004.

[69] Cynthia Dwork. “An ad omnia approach to defining and achieving private data analysis”. In: *PST in KDD*. 2008.

[70] Cynthia Dwork. “Differential privacy: A survey of results”. In: *TAMC*. 2008.

[71] Cynthia Dwork et al. “Boosting and differential privacy”. In: *FOCS*. 2010.

[72] Cynthia Dwork et al. “Calibrating Noise to Sensitivity in Private Data Analysis”. In: *TCC*. 2006.

[73] Cynthia Dwork et al. “Pan-Private Streaming Algorithms.” In: *ICS*. 2010.

[74] Cynthia Dwork et al. “Robust traceability from trace amounts”. In: *FOCS*. 2015.

[75] Cynthia Dwork et al. “The Algorithmic Foundations of Differential Privacy”. In: *Foundations and Trends in Theoretical Computer Science* (2014).

[76] Harrison Edwards et al. “Censoring representations with an adversary”. In: *ICLR*. 2016.

[77] Tariq Elahi et al. “PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks”. In: *ACM CCS*. 2014.

[78] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE transactions on information theory* (1985).

[79] Dumitru Erhan et al. “The Difficulty of Training Deep Architectures and the Effect of Unsupervised Pre-Training”. In: *AISTATS*. 2009.

[80] Úlfar Erlingsson et al. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response”. In: *ACM CCS*. 2014.

[81] European Commission. *General European Data Protection Regulation (GDPR)*. `http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2012:0011:FIN:EN:PDF`. 2012.

[82] Giulia Fanti et al. “Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries”. In: *PETS* (2016).

[83] Matthew Fredrikson et al. “Privacy in pharmacogenetics: An end-to-end case study of personalized warfarin dosing”. In: *USENIX Security*. 2014.

[84] Matt Fredrikson et al. “Model inversion attacks that exploit confidence information and basic countermeasures”. In: *ACM CCS*. 2015.

[85] Robin C Geyer et al. “Differentially Private Federated Learning: A Client Level Perspective”. In: *arXiv preprint 1712.07557* (2017).

[86] Ian Goodfellow et al. “Deep Learning”. MIT Press. 2016.

[87] Ian Goodfellow et al. “Generative adversarial nets”. In: *NIPS*. 2014.

[88] David Goulet et al. *Hidden-Service statistics Reported by Relays*. `https://research.torproject.org/techreports/hidden-service-stats-2015-04-28.pdf`. 2015.

[89] Thore Graepel et al. "ML confidential: Machine Learning on Encrypted Data". In: *ICISC*. 2012.

[90] Ishaan Gulrajani et al. "Improved training of wasserstein gans". In: *arXiv preprint 1704.00028* (2017).

[91] Jihun Hamm et al. "Learning privately from multiparty data". In: *ICML*. 2016.

[92] Moritz Hardt et al. "A multiplicative weights mechanism for privacy-preserving data analysis". In: *FOCS*. 2010.

[93] Moritz Hardt et al. "A Simple and Practical Algorithm for Differentially Private Data Release". In: *NIPS*. 2012.

[94] Jamie Hayes et al. "LOGAN: Membership Inference Attacks Against Generative Models". In: *Under Submission - arXiv preprint 1705.07663*. PoPETS 2019.

[95] Jonathan L Herlocker et al. "Evaluating Collaborative Filtering Recommender Systems". In: *ACM TOIS* (2004).

[96] Briland Hitaj et al. "Deep models under the GAN: information leakage from collaborative deep learning". In: *ACM CCS*. 2017.

[97] Susan Hohenberger et al. "ANONIZE: A large-scale anonymous survey system". In: *IEEE S&P*. 2014.

[98] Nils Homer et al. "Resolving individuals contributing trace amounts of DNA to highly complex mixtures using high-density SNP genotyping microarrays". In: *PLoS Genet* (2008).

[99] Chih-Chung Hsu et al. "CNN-Based Joint Clustering and Representation Learning with Feature Drift Compensation for Large-Scale Image Data". In: *IEEE Transactions on Multimedia* ().

[100] Justin Hsu et al. “Distributed Private Heavy Hitters”. In: *ICALP*. 2012.

[101] Chong Huang et al. “Context-aware generative adversarial privacy”. In: *Entropy* (2017).

[102] Gary B. Huang et al. *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments*. Tech. rep. `http://vis-www.cs.umass.edu/lfw/lfw.pdf`. University of Massachusetts, Amherst, 2007.

[103] Peihao Huang et al. “Deep embedding network for clustering”. In: *IEEE International Conference on Pattern Recognition (ICPR)*. 2014.

[104] Sergey Ioffe et al. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In: *ICML*. 2015, pp. 448–456.

[105] Geetha Jagannathan et al. “Privacy-preserving imputation of missing data”. In: *Data & Knowledge Engineering* (2008).

[106] Rob Jansen et al. “Safely measuring tor”. In: *ACM CCS*. 2016.

[107] Marek Jawurek et al. “Fault-Tolerant Privacy-Preserving Statistics”. In: *PETS*. 2012.

[108] Shouling Ji et al. “On Your Social Network De-anonymizablity: Quantification and Large Scale Evaluation with Seed Knowledge.” In: *NDSS*. 2015.

[109] Jinyuan Jia et al. “AttriGuard: A Practical Defense Against Attribute Inference Attacks via Adversarial Machine Learning”. In: *USENIX Security*. 2018.

[110] Kaggle.com. *Diabetic Retinopathy Detection*. `https://www.kaggle.com/c/diabetic-retinopathy-detection`.

[111] Andrej Karpathy et al. *Generative Models*. `https://blog.openai.com/generative-models/`. 2017.

[112] Tero Karras et al. “Progressive growing of gans for improved quality, stability, and variation”. In: *ICLR*. 2018.

[113] Emilia Käsper. “Fast Elliptic Curve Cryptography in OpenSSL”. In: *FC*. 2012.

[114] Yoon Kim. “Convolutional neural networks for sentence classification”. In: *arXiv preprint 1408.5882* (2014).

[115] Diederik P Kingma et al. “Auto-Encoding Variational Bayes”. In: *ICLR*. 2013.

[116] Diederik Kingma et al. “Adam: A method for stochastic optimization”. In: *arXiv preprint 1412.6980* (2014).

[117] Rob Kitchin. *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage, 2014.

[118] Naveen Kodali et al. “On convergence and stability of GANs”. In: *arXiv preprint 1705.07215* (2017).

[119] Alex Krizhevsky et al. *Learning multiple layers of features from tiny images*. Tech. rep. `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`. University of Toronto, 2009.

[120] Klaus Kursawe et al. “Privacy-friendly Aggregation for the Smart-grid”. In: *PETS*. 2011.

[121] Vasileios Lampos et al. “Advances in nowcasting influenza-like illness rates using search query logs”. In: *Scientific reports* (2015).

[122] Anders Boesen Lindbo Larsen et al. “Autoencoding beyond pixels using a learned similarity metric”. In: *arXiv preprint 1512.09300* (2015).

[123] Yann LeCun et al. “Deep Learning”. In: *Nature* (2015).

[124] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* (1998).

[125] Christian Ledig et al. *Photo-realistic single image super-resolution using a generative adversarial network*. 2016.

[126] Fengfu Li et al. “Discriminatively Boosted Image Clustering with Fully Convolutional Auto-Encoders”. In: *arXiv preprint 1703.07980* (2017).

[127] Haoran Li et al. “Differentially private synthesization of multi-dimensional data using copula functions”. In: *EDBT*. 2014.

[128] Ninghui Li et al. “t-closeness: Privacy beyond k-anonymity and l-diversity”. In: *IEEE ICDE*. 2007.

[129] Yujun Lin et al. “Deep Gradient Compression: Reducing the Communication Bandwidth for Distributed Training”. In: *ICLR*. 2018.

[130] Zi Lin et al. “Efficient Private Proximity Testing with GSM Location Sketches”. In: *FC*. 2012.

[131] Yehuda Lindell et al. “Privacy preserving data mining”. In: *CRYPTO*. 2000.

[132] Fang Liu. “Model-based differential private data synthesis”. In: *arXiv preprint 1606.08052* (2016).

[133] Jian Liu et al. “Oblivious neural network predictions via minionn transformations”. In: *ACM CCS*. 2017.

[134] Yunhui Long et al. “Understanding Membership Inferences on Well-Generalized Learning Models”. In: *arXiv preprint 1802.04889* (2018).

[135] Mario Lucic et al. “Are GANs Created Equal? A Large-Scale Study”. In: *arXiv preprint 1711.10337* (2017).

[136] Laurens van der Maaten et al. “Visualizing data using t-SNE”. In: *JMLR* (2008).

[137] Ashwin Machanavajjhala et al. “l-diversity: Privacy beyond k-anonymity”. In: *IEEE ICDE*. 2006.

[138] Ashwin Machanavajjhala et al. “Privacy: Theory meets practice on the map”. In: *ICDE*. 2008.

[139] Mohammad Malekzadeh et al. “Protecting sensory data against sensitive inferences”. In: *EuroSys Workshop on Privacy by Design in Distributed Systems*. 2018.

[140] M Malekzadeh et al. “Replacement autoencoder: A privacy-preserving algorithm for sensory data analysis”. In: *IEEE IoTDI*. 2018.

[141] Xudong Mao et al. “Least squares generative adversarial networks”. In: *IEEE ICCV*. 2017.

[142] David McClure et al. “Differential Privacy and Statistical Disclosure Risk Measures: An Investigation with Binary Synthetic Data.” In: *TDP* (2012).

[143] H Brendan McMahan et al. “Communication-efficient learning of deep networks from decentralized data”. In: *AISTATS*. 2017.

[144] H Brendan McMahan et al. “Learning differentially private language models without losing accuracy”. In: *ICLR*. 2018.

[145] F. McSherry. “Privacy integrated queries: An extensible platform for privacy-preserving data analysis”. In: *SIGMOD*. 2009.

[146] F. McSherry et al. “Mechanism design via differential privacy”. In: *FOCS*. 2007.

[147] Luca Melis et al. “Efficient private statistics with succinct sketches”. In: *NDSS*. 2016.

[148] Luca Melis et al. “Exploiting Unintended Feature Leakage in Collaborative Learning”. In: *IEEE S&P*. 2019.

[149] Luca Melis et al. “Private processing of outsourced network functions: Feasibility and constructions”. In: *ACM SDN-NFV Security*. 2016.

[150] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *NIPS*. 2013.

[151] Darakhshan Mir et al. “Pan-Private Algorithms via Statistics on Sketches”. In: *PODS*. 2011.

[152] Payman Mohassel et al. “SecureML: A system for scalable privacy-preserving machine learning”. In: *IEEE S&P*. 2017.

[153] Anna Monreale et al. “Privacy-Preserving Distributed Movement Data Aggregation”. In: *GISHE*. 2013.

[154] Philipp Moritz et al. “SparkNet: Training deep networks in Spark”. In: *arXiv preprint 1511.06051* (2015).

[155] Steven J Murdoch et al. “Low-cost traffic analysis of Tor”. In: *IEEE S&P*. 2005.

[156] Arvind Narayanan et al. “De-anonymizing social networks”. In: *IEEE S&P*. 2009.

[157] Arvind Narayanan et al. “How to break anonymity of the netflix prize dataset”. In: *arXiv preprint cs/0610105* (2006).

[158] Arvind Narayanan et al. “Location Privacy via Private Proximity Testing”. In: *NDSS*. 2011.

[159] Hong-Wei Ng et al. “A data-driven approach to cleaning large face datasets”. In: *ICIP*. 2014.

[160] Valeria Nikolaenko et al. “Privacy-Preserving Matrix Factorization”. In: *ACM CCS*. 2013.

[161] Brendan O’Connor et al. “From tweets to polls: Linking text sentiment to public opinion time series.” In: *ICWSM*. 2010.

[162] Seong Joon Oh et al. “Towards Reverse-Engineering Black-Box Neural Networks”. In: *ICLR*. 2018.

[163] Seyed Ali Ossia et al. “A Hybrid Deep Learning Architecture for Privacy-Preserving Mobile Analytics”. In: *ACM TKDD* (2018).

[164] otoro.net. *Generating Large Images from Latent Vectors*. `http://blog.otoro.net/2016/04/01/generating-large-images-from-latent-vectors/`. 2016.

[165] Jun Pang et al. “DeepCity: A feature learning framework for mining location check-ins”. In: *ICWSM*. 2017.

[166] Nicolas Papernot et al. “Scalable Private Learning with PATE”. In: *ICLR*. 2018.

[167] Nicolas Papernot et al. “Semi-supervised knowledge transfer for deep learning from private training data”. In: *ICLR*. 2017.

[168] Manas Pathak et al. “Multiparty differential privacy via aggregation of locally trained classifiers”. In: *NIPS*. 2010.

[169] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *JMLR* (2011).

[170] Jeffrey Pennington et al. “Spherical Random Features for Polynomial Kernels”. In: *NIPS*. 2015.

[171] NhatHai Phan et al. “Differential Privacy Preservation for Deep Auto-Encoders: an Application of Human Behavior Prediction”. In: *AAAI*. 2016.

[172] NhatHai Phan et al. “Preserving differential privacy in convolutional deep belief networks”. In: *ML* (2017).

[173] Raluca Ada Popa et al. “Privacy and accountability for location-based aggregate statistics”. In: *ACM CCS*. 2011.

[174] Apostolos Pyrgelis et al. “Knock Knock, Who’s There? Membership Inference on Aggregate Location Data”. In: *NDSS*. 2018.

[175] Jianwei Qian et al. “De-anonymizing social networks and inferring private attributes using knowledge graphs”. In: *INFOCOM*. 2016.

[176] Alec Radford et al. “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint 1511.06434* (2015).

[177] Ali Rahimi et al. “Random Features for Large-Scale Kernel Machines”. In: *NIPS*. 2007.

[178] Jerome P Reiter et al. “Bayesian estimation of disclosure risks for multiply imputed, synthetic data”. In: *JPC* (2014).

[179] Jerome P Reiter et al. “Estimating risks of identification disclosure in partially synthetic data”. In: *JPC* (2009).

[180] Paul Resnick et al. “Recommender Systems”. In: *Communications of the ACM* (1997).

[181] Marzieh Saeidi et al. “SentiHood: targeted aspect based sentiment analysis dataset for urban neighbourhoods”. In: *COLING*. 2016.

[182] Masaki Saito et al. “Temporal generative adversarial nets with singular value clipping”. In: *ICCV*. 2017.

[183] Tim Salimans et al. “Improved Techniques for Training GANs”. In: *NIPS*. 2016.

[184] Tim Salimans et al. “Weight normalization: A simple reparameterization to accelerate training of deep neural networks”. In: *NIPS*. 2016.

[185] Badrul Sarwar et al. “Item-based Collaborative Filtering Recommendation Algorithms”. In: *WWW*. 2001.

[186] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural networks* (2015).

[187] Bernhard Schölkopf et al. “Nonlinear Component Analysis as a Kernel Eigenvalue Problem”. In: *Neural Computation* 10.5 (1998).

[188] Elaine Shi et al. “Privacy-Preserving Aggregation of Time-Series Data”. In: *NDSS*. 2011.

[189] Reza Shokri et al. “Membership inference attacks against machine learning models”. In: *IEEE S&P*. 2017.

[190] Reza Shokri et al. “Privacy-preserving deep learning”. In: *ACM CCS*. 2015.

[191] Karen Simonyan et al. “Very deep convolutional networks for large-scale image recognition”. In: *arXiv preprint 1409.1556* (2014).

[192] Fabio Soldo et al. “Predictive blacklisting as an implicit recommendation system”. In: *INFOCOM*. 2010.

[193] Congzheng Song et al. “Machine Learning Models that Remember Too Much”. In: *ACM CCS*. 2017.

[194] Nitish Srivastava et al. “Dropout: a simple way to prevent neural networks from overfitting.” In: *JMLR* (2014).

[195] Dong Su et al. “Differentially Private K-Means Clustering”. In: *ACM CODASPY*. 2016.

[196] L. Sweeney. “$k$-Anonymity: A Model for Protecting Privacy”. In: *JUFKS* (2002).

[197] Lucas Theis et al. “A note on the evaluation of generative models”. In: *arXiv preprint 1511.01844* (2015).

[198] Lucas Theis et al. “Lossy image compression with compressive autoencoders”. In: *arXiv preprint 1703.00395* (2017).

[199] Tijmen Tieleman. “Training restricted Boltzmann machines using approximations to the likelihood gradient”. In: *ICML*. 2008.

[200] Jörn Tillmanns. “Privately Computing Set-Union and Set-Intersection Cardinality via Bloom Filters”. In: *ACISP*. 2015.

[201] Florian Tramèr et al. “Stealing machine learning models via prediction apis”. In: *USENIX Security*. 2016.

[202] Aleksei Triastcyn et al. “Generating Differentially Private Datasets Using GANs”. In: *arXiv preprint 1803.03148* (2018).

[203] Ardhendu Tripathy et al. “Privacy-Preserving Adversarial Networks”. In: *arXiv preprint 1712.07008* (2017).

[204] Stacey Truex et al. “Towards Demystifying Membership Inference Attacks”. In: *arXiv:1807.09173* (2018).

[205] Mohammed Tuhin. “Sanitization of Call Detail Records via Differentially-Private Bloom Filters”. In: *DBSec*. 2015.

[206] István Varga et al. “Aid is out there: Looking for help from tweets during a large scale disaster”. In: *ACL*. 2013.

[207] Sreekanth Vempati et al. “Generalized RBF feature maps for Efficient Detection”. In: *BMVC*. 2010.

[208] Ben Verhoeven et al. “CLiPS Stylometry Investigation (CSI) corpus: A Dutch corpus for the detection of age, gender, personality, sentiment and deception in text”. In: *LREC*. 2014.

[209] Binghui Wang et al. “Stealing Hyperparameters in Machine Learning”. In: *IEEE S&P*. 2018.

[210] Larry Wasserman et al. “A statistical framework for differential privacy”. In: *JASA* (2010).

[211] Xi Wu et al. “Differentially Private Stochastic Gradient Descent for in-RDBMS Analytics”. In: *arXiv preprint 1606.04722* (2016).

[212] Yuhuai Wu et al. “On the Quantitative Analysis of Decoder-Based Generative Models”. In: *ICLR*. 2017.

[213] Xiaokui Xiao et al. “iReduct: differential privacy with reduced relative errors”. In: *ACM SIGMOD*. 2011.

[214] Junyuan Xie et al. “Unsupervised Deep Embedding for Clustering Analysis”. In: *ICML*. 2016.

[215] Eric P Xing et al. “Petuum: A new platform for distributed machine learning on big data”. In: *IEEE TBD* (2015).

[216] Bo Yang et al. “Towards k-means-friendly spaces: Simultaneous deep learning and clustering”. In: *arXiv preprint 1610.04794* (2016).

[217] Dingqi Yang et al. “NationTelescope: Monitoring and visualizing large-scale collective behavior in LBSNs”. In: *JNCA* (2015).

[218] Dingqi Yang et al. “Participatory cultural mapping based on collective behavior in location based social networks”. In: *ACM TIST* (2015).

[219] Dingqi Yang et al. “PrivCheck: privacy-preserving check-in data publishing for personalized location based services”. In: *UbiComp*. 2016.

[220] Jianwei Yang et al. “Joint unsupervised learning of deep representations and image clusters”. In: *IEEE CVPR*. 2016.

[221] Raymond Yeh et al. “Semantic Image Inpainting with Perceptual and Contextual Losses”. In: *arXiv preprint 1607.07539* (2016).

[222] Samuel Yeom et al. “Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting”. In: *IEEE CSF*. 2018.

[223] Chiyuan Zhang et al. “Understanding deep learning requires rethinking generalization”. In: *ICLR*. 2017.

[224] Jun Zhang et al. “Functional mechanism: regression analysis under differential privacy”. In: *VLDB* (2012).

[225] Jun Zhang et al. “Privbayes: Private data release via bayesian networks”. In: *ACM TODS* (2017).

[226] Ning Zhang et al. “Beyond frontal faces: Improving person recognition using multiple cues”. In: *IEEE CVPR*. 2015.

[227] Yin Zheng et al. “Variational Deep Embedding: A Generative Approach to Clustering”. In: *IJCAI* (2017).

[228] Martin Zinkevich et al. “Parallelized stochastic gradient descent”. In: *NIPS*. 2010.