

Evolving Trade-offs Towards Deployable Private Systems for Data Science

by

Thomas Humphries

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2026

© Thomas Humphries 2026

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner: Prateek Saxena
Associate Professor, School of Computing
National University of Singapore

Supervisor: Florian Kerschbaum
Professor, Cheriton School of Computer Science
University of Waterloo

Internal Members: Xi He
Associate Professor, Cheriton School of Computer Science
University of Waterloo

Mohammad Hajiabadi
Assistant Professor, Cheriton School of Computer Science
University of Waterloo

Internal-External Member: Mahesh Tripunitara
Professor, Electrical and Computer Engineering
University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This thesis includes co-authored and peer-reviewed work that has appeared in conference proceedings.

Chapter 2 of this thesis was coauthored with Abdulrahman Diaa, Lucas Fenaux, Marian Dietz, Faezeh Ebrahimiaghazani, Bailey Kacsmar, Xinda Li, Nils Lukas, Rasoul Akhavan Mahdavi, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum [54]. Abdulrahman, Lucas and I were co-first authors of the publication. I was the project lead, responsible for overseeing the design, and writing all parts of the paper. Abdulrahman focused on the experiments and design of the cryptography portions of the paper. Lucas focused on the experiments and design of the machine learning parts of the paper. Marian, Faezeh, Xinda, Nils, and Rasoul contributed to an early version of the experiments for the paper. Bailey, Simon, Rasoul, and Ehsan took part in discussions and proofreading of the paper. Florian supervised the work.

Chapter 3 of this thesis was coauthored with Rasoul Akhavan Mahdavi, Shannon Veitch, and Florian Kerschbaum [102]. Rasoul, Shannon, and I were joint first authors of the publication who all actively contributed to all aspects of the paper. I was responsible for the initial idea and the privacy proofs. Rasoul focused on the experimental evaluation. Shannon focused on the protocol definitions and security proofs. Florian was the supervisor of the project.

Chapter 4 of this thesis was coauthored with Zinan Lin and Sergey Yekhanin [105]. I was the first author of the work. I was responsible for the design, writing, and experimentation. Zinan and Sergey contributed through project discussions and proofreading of the work, and Sivakanth Gopi also contributed to discussions in the early stages of the project.

Chapter 5 of this thesis was coauthored with Tim Li, Shufan Zhang, Karl Knopf, and Xi He [104]. I was the first author of this paper. I designed, wrote, and proved all parts of the algorithm, as well as oversaw the experimental work. Tim was responsible for all experimental evaluation. Shufan wrote the experimental section, and Karl wrote the related work section. Shufan and Karl also participated in discussions and proofreading of the paper. Xi supervised the project.

Chapter 6 of this thesis was coauthored with Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum [106]. I was the first author of the paper. I worked on all parts of the paper, focusing on the experiments and writing. Simon was responsible for the theory and proofs of the paper, while also helping with experiments and writing. Lindsey, Matthew, and I worked on an earlier course project

version of the work, where Matthew focused on experiments and Lindsey on writing. Ian and Urs discussed and proofread the paper. Florian supervised the work.

Abstract

There is no one-size-fits-all solution to preserving privacy in data science. While insights derived from sensitive data can benefit society, privacy is typically at odds with utility, performance, usability, or some combination of these objectives. Furthermore, each system differs in its definition of these objectives and the way they interact with one another. If the compromise required for any single objective is too great, the system will not be deployed, or worse, will be deployed with a weakened privacy guarantee, exposing users to potential harm. In this work, we address this challenge from multiple angles. First, through strategic algorithm design, our work creates private systems with improved trade-offs, enabling their deployment. This includes a more efficient protocol for the secure inference of deep machine learning models, a novel construction for aggregating key-value data in the local trust model, an evolutionary approach to improve the utility of private clustering, and a user-friendly interpretation of the error of private median queries. Second, we audit private systems to show the privacy risks associated with misleading privacy claims. In particular, through a privacy audit of machine learning, we highlight a difference between expectation and reality in privacy protections.

Acknowledgements

First, a huge thank you to my supervisor, Florian. I appreciated your constant support without ever applying pressure, the subtle way you gave hints so that I would learn the lesson myself, and your encouragement to take opportunities that were best for me regardless of whether they were beneficial to you. I hope I can mirror these qualities as a supervisor.

Thanks to my committee members Xi He, Mohammad Hajiabadi, Mahesh Tripunitara, and Prateek Saxena for sharing your time and insights.

I have been fortunate to have such incredible mentors and friends: Bailey, Simon, and Miti. I truly appreciate you all being my 24/7 support line for any issue, big or small. Working on projects with each of you inspired me and shaped my approach to research.

I want to thank all my wonderful collaborators, especially Xi, Zinan, and Sergey, for all your support that extended far beyond the projects we worked on.

This dissertation would not have been possible without my amazing friends and colleagues in the CrySP lab. The CrySP lab is a genuinely special place that goes beyond a friendly lab culture to a large family. I have made lifelong friends during my time here and hope we all stay in touch wherever we end up. I have thoroughly enjoyed getting to know so many wonderful people throughout my time in the lab. While I would like to name everyone, that list is simply too long. However, I would be remiss not to name the people who I made the most memories with: John, Rasoul, Matt, Lucas, Shannon, Diaa, Jason, and Pranshu. I am incredibly lucky to have such sincere friends.

Finally, to my family. Thanks for standing by my side through my procrastination-fuelled marathon of an academic journey. You were always there for me, whether through long discussions or rearranging things to make space for my studies. You always knew I could do it, even when I wasn't so sure myself.

Dedication

To my selfless wife Shane and adorable daughter Claire: your endless support means everything.

Table of Contents

Examining Committee Membership	ii
Author’s Declaration	iii
Statement of Contributions	iv
Abstract	vi
Acknowledgements	vii
Dedication	viii
List of Figures	xv
List of Tables	xix
1 Introduction	1
2 Fast and Private Inference of Deep Neural Networks by Co-designing Activation Functions	4
2.1 Introduction	4
2.2 Background	7
2.2.1 Multi Party Computation	7
2.2.2 Neural Network Inference	7

2.3	Problem Setup and Motivation	8
2.3.1	Problem Setup	8
2.3.2	Privacy During Model Training	9
2.3.3	Motivating the Co-Design of Activation Functions	9
2.4	Faster Evaluation of Polynomials	10
2.4.1	The Polynomial Advantage	10
2.4.2	ESPN: <u>E</u> xponentiating <u>S</u> ecret Shared Values using <u>P</u> ascal’s tri <u>a</u> ngle	12
2.4.3	Polynomial Evaluation with ESPN	13
2.4.4	Correctness Proofs	15
2.4.5	Alternative Single Round Protocol: HoneyBadger	17
2.5	PILLAR: <u>P</u> olynomial <u>A</u> ctivation <u>R</u> egularization	19
2.5.1	The Problem with Polynomial Activation Functions	19
2.5.2	Defining PILLAR	20
2.5.3	Measuring PILLAR’s Effectiveness	24
2.6	Evaluation of Co-Design	25
2.6.1	Experimental Setup	26
2.6.2	ResNet-18 Architecture	27
2.6.3	VGG-16 Architecture	29
2.6.4	Other Architectures	31
2.6.5	Scaling to ImageNet	32
2.6.6	Real WAN Evaluation	33
2.7	Communication and Rounds Benchmark	34
2.8	Discussion	35
2.9	Related Work	36
2.10	Conclusion	37

3	Selective MPC: Distributed Computation of Differentially Private Key-Value Statistics	38
3.1	Introduction	38
3.2	Preliminaries	41
3.2.1	Differential Privacy	41
3.2.2	Secret Sharing	42
3.2.3	Secure Multi-party Computation	42
3.3	Problem Statement	43
3.3.1	Notation	43
3.3.2	Assumptions	44
3.3.3	System Architecture	46
3.4	Preliminary Solutions	47
3.5	Protocol Description	49
3.5.1	Data Collection Phase	49
3.5.2	Multi-party Computation Phase	50
3.5.3	Implementation Considerations	53
3.6	Privacy	54
3.6.1	Privacy of the Output	59
3.7	Security	61
3.7.1	Proof of security	61
3.7.2	Extending to the malicious model.	62
3.8	Evaluation	64
3.8.1	Our Protocol	64
3.8.2	Comparison with Other Work	66
3.8.3	Experimental Results	68
3.9	Discussion	70
3.10	Related Work	73
3.11	Conclusion	75

4	PE-means: Improved Differentially Private k-means Clustering through Private Evolution	76
4.1	Introduction	76
4.2	Background	78
4.2.1	k -Means Clustering	78
4.2.2	Differential Privacy	78
4.3	Methodology	80
4.3.1	Random API	81
4.3.2	Selection Step	81
4.3.3	Variation API	83
4.3.4	Adaptive Population Size	84
4.3.5	High-Dimensional Case	84
4.3.6	Privacy Analysis	86
4.4	Experiments	87
4.4.1	Experimental Setup	88
4.4.2	Vote Splitting Example	90
4.4.3	Hyperparameter Tuning	90
4.4.4	Utility Benchmark	93
4.4.5	Discussion	96
4.5	Related Work	97
4.6	Conclusion	99
5	Interpreting the Error of Differentially Private Median Queries through Randomization Intervals	101
5.1	Introduction	101
5.2	Background	103
5.2.1	Problem Setup	103
5.3	PostRI	104

5.3.1	Median Mechanism	104
5.3.2	RI Mechanism Design	105
5.3.3	Privacy Analysis	107
5.3.4	Correctness	108
5.4	Utility Analysis	109
5.4.1	Hyperparameter Selection	111
5.5	Preliminary Experimental Results	112
5.6	Related Work	115
5.7	Concluding Remarks	115
6	Investigating Membership Inference Attacks under Data Dependencies	116
6.1	Introduction	116
6.2	Preliminaries	119
6.2.1	Membership Inference Attacks	120
6.2.2	Differential Privacy in Machine Learning	121
6.3	Membership Experiments	121
6.3.1	Strong-Adversary Membership Experiment	122
6.3.2	IID Membership Experiment	124
6.3.3	A Membership Experiment with Data Dependencies	126
6.4	Empirical Evaluation of Membership Inference with Data Dependencies . .	131
6.4.1	Evaluation Setup	131
6.4.2	Cluster-based Dependencies	134
6.4.3	Dependencies Caused by an Attribute Bias	137
6.4.4	Inherent Dataset Dependencies	139
6.5	Discussion: Connection to Property Inference	143
6.6	Related Work	145
6.7	Conclusion	147

7 Conclusion	148
References	151
Appendices	172
A Privacy of the One-Sided Dummy Solution	173
A.1 Background on the Geometric Mechanism	173
A.2 Privacy proof	174
A.3 How many dummies to add	175
A.4 Comparison with our protocol	175
A.5 Using the noise from dummies	176

List of Figures

2.1	Summary of the inference time in seconds vs. test accuracy for each state-of-the-art approach on the CIFAR-10 dataset in the WAN (100 ms roundtrip delay).	5
2.2	Benchmarking the secure evaluation of ReLU activation functions using various approaches. The x -axis is the network delay in ms and the y -axis is the mean runtime in seconds averaged over 20 runs with the shaded area representing the 95% confidence intervals.	11
2.3	Polynomial Input Distribution of ResNet18 on CIFAR10.	14
2.4	Accuracy of a 2-layer convolutional network trained with varying degrees for the polynomial activation function.	18
2.5	Illustrating the escaping activation problem for the two layers convolutional network.	18
2.6	The effect of the regularization coefficient, β on model accuracy and out-of-range ratio for $\gamma = 10$	18
2.7	The effect of truncation on a polynomial activation function.	20
2.8	Summary of the inference time vs accuracy for each state-of-the-art approach on the CIFAR-100 dataset in the WAN (100 ms roundtrip delay).	25
2.9	ResNet-18 evaluation on CIFAR-100 (20 runs).	28
2.10	VGG-16 evaluation on CIFAR-100 (20 runs).	29
2.11	Evaluating the various COINN architectures (20 runs).	30
2.12	ImageNet evaluation on ResNet-50 (20 runs).	33
3.1	Data flow.	46

3.2	Client process for a single key-value pair.	50
3.3	We visualize the distribution of a single computation node's view of 10 keys. We set $p = 2/3$ and $r = 0.4$ and plot the distribution with and without dummies.	54
3.4	ϵ_L for varying levels of collusion.	59
3.5	Run time for a single key. Measurements are means with 95% confidence interval in shaded region. ϵ_L is shown for the number of computation nodes.	67
3.6	Communication for a single key. Measurements are means with 95% confidence interval in shaded region. ϵ_L is shown for the number of computation nodes.	67
3.7	(a) Leakage privacy as a function of r for $t = 2$. (b) Lower bound on leakage privacy.	73
4.1	A visualization of the private data, samples generated, and samples selected by PE in each iteration, comparing the previous top- k selection approach to our proposed k -means approach with $\epsilon = \infty$. The k -means approach effectively mitigates the vote splitting problem and keeps selected points close to the true clusters.	89
4.2	A scatter plot of the top-3 parameter configurations of L and T for each dataset at $\epsilon = 1$. The LOWESS line of best fit shows a clear relationship between L and N and between T and d that are captured by our heuristics.	91
4.3	A scatter plot of grid search results over L and T with $\epsilon = 1$. There is one point per dataset with the x-axis shows the clustering loss of the optimal parameter setting in the grid and the y-axis showing the loss of our heuristic for that dataset. We observe that the loss of our heuristic parameter settings are close to the optimal loss for all datasets.	93
4.4	A heat map of the grid search results over γ and β at $\epsilon = 1$ with each cell showing the percentage increase in clustering loss for a given hyperparameters setting compared to the optimal hyperparameter choice for each dataset, averaged over all datasets. We observe our default parameter choice is the closest to the optimal loss.	94

4.5	A scatter plot of grid search results over γ and β with $\epsilon = 1$. There is one point per dataset with the x-axis shows the clustering loss of the optimal parameter setting in the grid and the y-axis showing the loss of our default parameter values. We observe that the loss of our default parameter settings are close to the optimal loss for all datasets.	95
4.6	A plot of the privacy-utility trade-off of all approaches on real datasets, showing a PE-based approach always performs best.	96
4.7	A plot of the label accuracy metric vs. privacy budget ϵ over all approaches on a subset of real datasets that have ground-truth labels. We observe that a PE-based approach always performs better or similar to state-of-the-art.	97
4.8	A comparison illustrating how approaches scale with the number of clusters k over different synthetic datasets at $\epsilon=1.0$	98
5.1	Average median error and RI length vs. varying privacy budget on the Banking dataset	114
5.2	Average median error and RI length vs. varying privacy budget on the Adult dataset	114
5.3	Average median error and RI length vs. varying privacy budget on the Airplane dataset	114
6.1	Upper bounds on the membership advantage in the IID Membership Experiment.	126
6.2	Results in <code>adult</code> dataset with cluster-based dependencies.	135
6.3	Results in <code>compas</code> dataset with cluster-based dependencies.	135
6.4	Results in <code>adult</code> dataset with gender bias (p determines the proportion of males in the member set).	138
6.5	Results in <code>adult</code> dataset with education bias (p determines the proportion of individuals with low education in the member set).	138
6.6	Performance of the optimal threshold attack in the <code>heart</code> dataset when members belong to a particular database (hospital/institution), and non-members are taken from all other databases.	140
6.7	Results in <code>students</code> dataset when members are from the Gabriel Pereira school, and non-members from Mousinho da Silveira school.	141

6.8	Results in texas dataset when members are from hospitals in region 3, and non-members are from any other region.	142
6.9	Results when training a model with census , where non-members are from adult dataset.	143
A.1	We consider the expected number of dummies for various epsilons. r is chosen to minimize the epsilon spent and where applicable delta is fixed to $\delta = 2^{-40}$.	176

List of Tables

2.1	Comparing the effect of BatchNorm on MiniONN model.	23
2.2	Plain-text Accuracy of PILLAR (5 runs).	24
2.3	ResNet-18 accuracy comparison (5 runs).	29
2.4	VGG-16 accuracy comparison (5 runs).	31
2.5	Accuracy comparison on the various architectures considered in COINN (5 runs).	32
2.6	ImageNet accuracy comparison (1 run).	33
2.7	Real World WAN evaluation. We report total inference time in seconds. . .	34
2.8	Evaluation of communication in GB and the number of rounds (shown in parenthesis).	34
3.1	Notation	44
3.2	Comparison of client-server communication, computation (number of multiplications), and error (approximate bound). *Includes a multiplicative factor equal to the number of iterations which we exclude.	66
3.3	Run time (in seconds) for one key. Measurements are means with standard deviations in parentheses. For each configuration the first row and second row use a semi-honest and maliciously secure MPC, respectively.	71
3.4	Run time (in seconds) for many keys with 5 computation nodes, which corresponds to $\epsilon_L = 0.758$. Measurements are means with standard deviations in parentheses. For each configuration the first and second row use a semi-honest and maliciously secure MPC, respectively.	71

3.5	Run time (in seconds) to validate multiple inputs with 3 computation nodes using maliciously secure MPC. Measurements are means with standard deviations in parentheses.	72
4.1	A summary of results over all datasets (real datasets highlighted in bold) showing the AUC of the clustering loss across ϵ values with the best approach highlighted for each dataset. The final column shows the relative improvements of PE-based techniques over the state-of-the art baseline for that dataset, with an average improvement of 26% over all datasets for PE-based approaches.	100
5.1	Comparison of Median RI on Real Datasets with $\epsilon = 1$ and $\beta = 0.01$	112
6.1	Notation	119
6.2	Evaluation Setup Summary	131

Chapter 1

Introduction

The best way to truly protect one’s privacy is to not share any sensitive information. However, in today’s data-driven society, not sharing data means not participating in many aspects of daily life. Organizations are storing colossal amounts of data about their users, which can lead to favourable outcomes, if they take the necessary steps to protect privacy. For example, in health care, the vast amounts of data stored in digital health records can be of great use to the medical research community. Another example is computing statistics over product usage data to inform the development of more useful digital services. However, classic data science algorithms, without modification, are vulnerable to a wide range of privacy attacks [12, 34, 60, 93, 205]. Naive privacy modifications such as removing personally identifiable information or heuristic attempts at privacy based on similarity have also been found to be vulnerable to attack [144, 160].

In response to this problem, many effective privacy-preserving solutions have been deployed. Secure multi-party computation (MPC) allows a set of parties to jointly compute a function while keeping the input private and revealing only the output. Multi-party computation has been deployed to collect statistics about the wage gap in the U.S. [125] or to collect aggregate usage information from the exposure app during COVID-19 [16]. MPC is also becoming increasingly popular with organizations such as Apple [16, 189], Google [16], and Let’s Encrypt [9]. Another increasingly popular approach to utilizing private data while maintaining the privacy of individuals is differential privacy (DP) [69]. The mathematical definition of differential privacy guarantees that the information learned from a private database would be approximately the same with or without any single user’s participation. Using this notion, data analysts can perform many useful queries on the data while giving an intuitive privacy guarantee to participants. For example, DP has been deployed to protect the U.S. Census from reconstruction attacks [12] as well as to privately

report statistics about the effectiveness of social distancing protocols during COVID-19 [14]. Many organizations such as Apple [56], Google [14, 75], and Microsoft [57] have adopted differential privacy to protect user data.

The challenge is that privacy-preserving systems add significant overheads in order to preserve privacy. Making an algorithm private often degrades its utility, performance, usability, or some combination of these goals. To encourage an organization to switch from a non-private to a private version, the trade-off must be reasonable. For example, if the protocol used to collect usage data from the COVID-19 exposure app had drained the batteries of user’s phones, it would not have been deployed. Similarly, if the DP noise added to the U.S. Census had made the statistics unusably inaccurate, the privacy protections may have been weakened as the census is essential to many organizations. The reality is that while there have been numerous success stories for the deployment of private systems, there are even more cases where we do not have an easily deployable private alternative. Motivated by this challenge, in this thesis, we ask the following research question:

How can we design deployable data science systems with meaningful privacy guarantees?

In response to this research question, we first argue why a per-system approach to privacy-preserving design leads to more deployable systems. A core reason is that the specific trade-off preventing deployment varies per-system. At a high level, we can group these trade-offs into categories such as performance, utility, and usability. However, these terms only make sense when defined within the specific context of a system. Measuring utility depends on the assumed use case of the system’s output; performance metrics depend on the assumed compute; and usability depends on the user base’s background knowledge. Furthermore, the applicable tools and techniques to improve these trade-offs vary based on the requirements of the system. For example, is it reasonable to assume a party has the resources to run cryptographic protocols or train a large machine learning model? We clarify that this does not mean that there are no transferable techniques across systems, but even transferability is system-dependent. In support of the effectiveness of per-system design, we present multiple case studies of systems with unique privacy trade-offs and present tailored solutions that significantly increase the deployability of the private alternative. In Chapter 2, we address the performance trade-off in secure machine learning (ML) inference through designing new MPC and ML techniques. In Chapter 3, we redefine the bottleneck of key-value data analytics from utility to performance, which we then address through the co-design of DP and MPC. In Chapter 4, we improve the privacy-utility trade-off of k -means clustering by designing an evolutionary algorithm that can effectively optimize

cluster centres while using less information about the private data. Finally, in Chapter 5, we improve the usability trade-off of computing the median privately by including interpretable bounds on the noisy median without degrading utility.

To address the second part of our research question, we expand on what constitutes a meaningful privacy guarantee and on our approach to ensuring privacy claims are well-founded. We argue that meaningful privacy guarantees are those that offer formal proofs of privacy. However, privacy is inherently individualized and difficult to define. Thus, a proof is only effective when its definitions match the system requirements established by the various stakeholders. Only the stakeholders themselves can accurately make this determination, but we argue that as researchers, we can offer experimental evidence to better communicate the advantages and disadvantages of different definitions. This evidence comes from conducting privacy audits. Auditing consists of running privacy attacks against a system to study the empirical privacy protection offered in various scenarios. Audits can be useful for many reasons. A system may be theoretically sound, but the implementation may have a bug violating privacy that is uncovered during an audit. Alternatively, privacy audits can uncover a mismatch in expectations from the theoretical model assumed and the actual privacy risks encountered in practice. We give one such example of a mismatch in privacy expectations vs. practical privacy attacks in Chapter 6 where we audit private ML.

Chapter 2

Fast and Private Inference of Deep Neural Networks by Co-designing Activation Functions

This chapter is adapted from work that previously appeared at the USENIX Security Symposium [54]. For this case study, we consider clients with privacy-sensitive inputs on which they wish to obtain predictions from a machine learning classifier. This system assumes that clients do not have the resources or desire to host a model themselves. Rather, they wish to use a machine learning service. Ensuring the input remains private is key to protecting the client’s input and avoiding the liability of data breaches caused by the service provider. The challenge is that existing solutions to conduct secure inference over a machine learning model are prohibitively slow and do not scale to deeper architectures. In this chapter, we make significant progress in designing a system with reasonable run times for deeper machine learning models.

2.1 Introduction

The rapid development of increasingly capable machine learning (ML) models has resulted in significant demand for products like machine learning as a service (MLaaS). In this scenario, big tech companies with vast computing resources train large machine learning models and provide users with query access. The major pitfall with MLaaS is that it requires clients to submit potentially sensitive queries to an untrusted entity. A promising

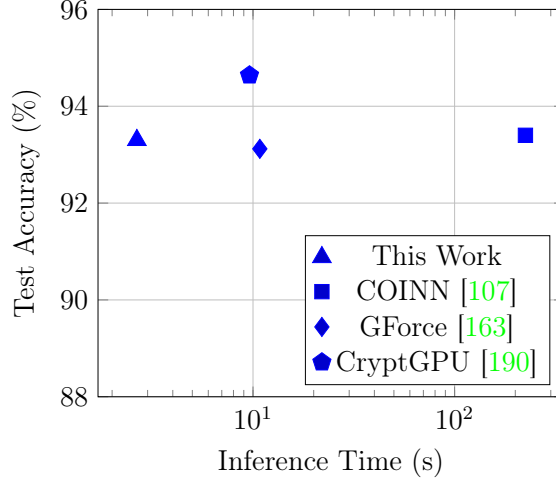


Figure 2.1: Summary of the inference time in seconds vs. test accuracy for each state-of-the-art approach on the CIFAR-10 dataset in the WAN (100 ms roundtrip delay).

solution to this problem is to employ cryptography to ensure the queries and inferences are hidden from the model owner. Secure inference is an active field of research with many solutions and different threat models as summarized in a recent SoK [164]. The challenge is that despite recent advances, the inference times are still prohibitively large compared to plaintext inferences.

This work focuses on reducing the runtime of secure inference on image data, under realistic network conditions, while maintaining classification accuracy. We consider the two-party setting using multi-party computation (MPC), where the server holds the modified ML model, and the client holds the data to query the model. Recent state-of-the-art works in this space employ various co-design approaches to reduce the inference time [164]. For example, COINN co-designs ML models optimized for quantization with efficient MPC protocols tailored to the custom models [107]. COINN substantially compresses the model and makes numerous optimizations to the architecture to achieve fast inferences. Another example is GForce, which tailors the cryptography needed for ML to high-speed GPU hardware [163]. By offloading vast amounts of work to the pre-computation phase, they are able to achieve state-of-the-art runtime and accuracy in secure inference [163]. Similarly, CryptGPU [190] modifies the CrypTen framework [121] to run efficiently on the GPU and give state-of-the-art inference times in wide area networks. However, despite making major steps towards practical inference, none of these works remove a crucial bottleneck in secure inference: the non-linear layers.

It is well known that the non-linear layers are the bottleneck of secure inference [83, 85, 107, 152]. This is because secure computation on arithmetic shares is optimized for multiplications and additions, instead of non-linear layers such as ReLU activation functions or MaxPool layers. In order to compute these non-linear functions, expensive conversions between different types of MPC protocols are required. Specifically, in more realistic network settings with high latency, the inference time is substantially degraded due to each conversion taking many rounds of communication. This problem is particularly prevalent in deep neural networks (DNNs), where a non-linear activation separates each of the many linear layers.

This work addresses the non-linear layers by taking a co-design approach between the activation functions and MPC. We take the approach of replacing classic ReLU activation functions with a polynomial approximation during training and inference to avoid conversions altogether. Previous work has considered this approach but with limited success [83]. We propose two modifications to make this approach practical. First, we develop and evaluate new single-round MPC protocols that give the fastest evaluation of polynomials to date. The challenge with using polynomials is they severely impact model accuracy [83, 107, 152]. Previous work could not successfully train DNNs with more than 11 layers due to exploding gradients [83]. Thus, our second contribution is tailoring the ML training process to ensure high accuracy and stable training using polynomials. Our approach utilizes a new type of regularization that focuses on keeping the input to each activation function within a small range. We achieve close to plaintext accuracy on models as deep as ResNet-110 [97] and as large as a ResNet-50 on ImageNet [53] (23 million parameters). The combination of these approaches yields a co-design with state-of-the-art inference times and the highest accuracy for polynomial models.

We compare our work with three solutions representing the state-of-the-art approaches in secure inference according to Ng and Chow [164]. We summarize our results in Figure 2.1. Combining the single-round MPC protocols with our activation regularization achieves significantly faster inference times than all other solutions. Specifically, our solution is faster than CryptGPU by $4\times$, GForce by $5\times$, and COINN by $40\times$ on average in wide area networks. Our approach also scales to large models on ImageNet with a $110\times$ speedup over COINN, $14\times$ speedup over Cheetah, and a $3\times$ speedup over CryptGPU. Furthermore, our inference accuracy remains competitive with all other solutions. CryptGPU often gives slightly higher accuracy as it can evaluate any plaintext model (albeit slower than our work). Thus, the challenge for future work is to further close the ML accuracy gap between plain and polynomial models.

2.2 Background

2.2.1 Multi Party Computation

Secure multi-party computation (MPC) allows a set of parties to jointly compute a function while keeping their inputs to the function private. We focus on a variant of MPC which performs operations over shares of the data [28]. We use $[[s]] = [[s]_A, [s]_B]$ to denote that the value of s is shared among participants, where $[s]_A$ is the share held by party A and $[s]_B$ by party B . Arithmetic MPC protocols utilize a linear secret sharing scheme, such as an additive secret sharing scheme to compute complex circuits using combinations of additions and multiplications. Given constants v_1, v_2, v_3 and shares of values $[[x]], [[y]]$, one can locally compute

$$v_1[[x]] + v_2[[y]] + v_3 = [[v_1 \cdot x + v_2 \cdot y + v_3]] \quad (2.1)$$

to obtain shares of the value $v_1 \cdot x + v_2 \cdot y + v_3$. For multiplication, one can use Beaver’s trick to multiply using a single round of communication between parties [27]. Specifically, we assume a triplet of random numbers a, b, c (called a Beaver triplet) was generated such that $a \cdot b = c$ and secret shared among all parties ahead of time (typically in an offline precomputation phase). Then the parties compute $[[x \cdot y]]$ by first locally computing $[[a + x]] = A$ and $[[b + y]] = B$ and reconstructing A and B so that both parties have them in plaintext. This reconstruction is the single round required. Using these values, the parties compute the result locally as $[[x \cdot y]] = A[[y]] + (-B)[[a]] + [[c]]$, using the linearity property in equation 2.1.

Arithmetic MPC protocols are limited to basic multiplications and additions. Thus, for computing non-linear operations such as comparisons, other techniques such as converting to binary secret shares or using Yao’s garbled circuits are common [52]. A binary secret sharing scheme is an arithmetic scheme carried out bitwise in the ring $\mathbb{Z}_2$. Specifically, the difference is that we first decompose x into its bits and have a separate arithmetic share of each bit. By maintaining this bitwise structure, operations such as XOR or bit shifts are trivial. We describe how to use a binary and arithmetic secret-sharing scheme together to compute non-linear functions in Section 2.3.3.

2.2.2 Neural Network Inference

We consider DNN classifiers with domain $\mathcal{X} \subseteq \mathbb{R}^d$ and range $\mathcal{Y} \subseteq \mathbb{R}^c$. DNN classifiers consist of a sequence of layers, each performing either a linear or a non-linear operation. The

ResNet [97] architecture we consider is composed of (i) convolutional, (ii) fully connected, (iii) pooling, (iv) batch normalization, and (v) ReLU layers. All layers are linear, except for $\text{ReLU}(x) = \max(x, 0)$ and max pooling (that can be replaced with average pooling). To classify an input x , a classifier h , passes the input sequentially through each layer. Upon reaching the last layer, the prediction is obtained by taking $\arg \max_{i \in \{1..c\}} h(x)_i$, where we call h the logit function for a classifier $h : \mathcal{X} \rightarrow \mathcal{Y}$. Our work is tailored to securely perform the inference phase. Specifically, for an encrypted input x we compute the encrypted output $h(x)$ of the logit function. However, to achieve improvements during inference, our work requires modification to the training phase that generates h .

2.3 Problem Setup and Motivation

2.3.1 Problem Setup

We follow the same threat model as prior work for two-party secure inference [107, 121, 152, 163]. Specifically, we follow the two-party client-server model where the server has a machine learning model (a DNN) they have trained (with our technique), and the client holds private data upon which they would like to make an inference. Like Delphi [152], our work assumes the server uses a modified training procedure. The server’s input to the protocol is the weights of their trained model, which they do not want to leak to the client (due to intellectual property or protecting their MLaaS business [192]). The client has a private input (typically an image) they would like to classify using the model but do not want to leak this input or the prediction to the server. That is, the MPC function can be written as $f(\text{image}, \text{model}) = (\text{label}, \emptyset)$. Following previous work [107, 121, 152, 163], we consider the semi-honest model [90, §7.2.2], where adversaries do not deviate from the protocol but may gather information to infer private information. Also, in line with previous work, we assume the model architecture is known to both parties. This includes the dimensions and type of each layer, parameters such as field size used for inference, and the mean and standard deviation of the training set [121].

Our goal is to reduce the inference time as much as possible. We are willing to incur a small degradation in accuracy to achieve practical runtimes. For example, in the streaming setting, applications like spam detection are runtime-critical (a small accuracy trade-off can be tolerated to make it feasible). Since all the protocols we compare with contain pre-computation, we focus on the online phase for fair comparison. Furthermore, the online time determines the latency, which we focus on reducing in this work. We build from CrypTen [121], which does not implement pre-computation and rather assumes a third-party

dealer for Beaver triplets. However, in practice, the server and client could generate the Beaver triplets in a pre-computation phase using off-the-shelf protocols [118, 173].

2.3.2 Privacy During Model Training

We focus only on the inference phase of machine learning. However, the privacy of the training process and training data is an orthogonal but essential problem. We recommend that the data owner take appropriate steps to protect the privacy of the model, such as training using differential privacy [10] or rounding the output of the inference. Furthermore, during training, care should be taken to protect against threats such as model stealing, which can be launched using only the inference result [109]. To summarize, the model owner learns nothing other than the fact a query was made. We ensure only the inference is revealed to the client; however, ML attacks that only require black box query access [109] must be defended against during the training process. Since we focus the effects of MPC on the runtime and accuracy of ML inference, we did not conflate this comparison with additional privacy preserving training goals. Any privacy-preserving technique would add a similar overhead (e.g., reducing the accuracy) to all approaches we evaluate.

2.3.3 Motivating the Co-Design of Activation Functions

It has been well established in the literature that activation functions such as ReLU are the bottleneck in MPC-based secure inference, taking up to 93% of the inference time [83, 85, 107, 152]. The reason for this is that current approaches use different types of MPC protocols for a model’s linear and non-linear layers [107, 113, 121, 152, 163]. The linear layers are typically computed using standard arithmetic secret-sharing protocols tailored for additions and multiplications. The non-linear layers are computed using garbled circuits or binary secret share-based protocols. The bottleneck in wide area networks is typically the conversions between these protocols as they require a large number of communication rounds. A typical DNN architecture has many linear layers, each followed by a non-linear layer resulting in a prohibitively large number of conversions.

Consider CrypTen, a PyTorch-based secure ML library, as a baseline approach [121]. CrypTen uses binary shares to evaluate boolean non-linear layers such as ReLUs and MaxPooling layers. Specifically, all linear layers are computed using standard multiplication and addition protocols over arithmetic shares. To compute $[[ReLU(x)]]$ at each layer, $[[x]]$ is first converted to binary shares using a carry look-ahead adder. Once in binary shares, CrypTen extracts the sign bit to compute $[[x > 0]]$ (a local operation). The sign bit,

$[[x > 0]]$, is then converted back to arithmetic shares (trivial for a single bit) and multiplied with $[[x]]$ to get $[[ReLU(x)]]$. The problem with this approach is that each conversion takes $O(\log(L))$ communication rounds. Taking into account the additional round needed for multiplication, we observe nine communication rounds per ReLU in practice (under 64-bit precision). Recently, more sophisticated MPC protocols have been proposed that reduce the number of rounds needed for comparisons in arithmetic shares [35, 36] or reduce the cost of binary share conversions [77]. However, even if one were to implement these protocols in CrypTen, the number of rounds needed for non-linear layers would still outweigh the number needed for linear layers.

Motivated by this bottleneck, several works have focused on either reducing the number of ReLUs or replacing ReLUs altogether [83, 85, 113, 128, 152, 154]. One approach is to approximate each ReLU with a high degree polynomial [83, 128]. The advantage of using polynomials is that polynomials can be computed using arithmetic shares, thus removing the need for expensive conversions and improving the total inference time. A significant challenge with polynomials is maintaining model accuracy [83, 85, 107, 113, 152].

Thus, this work aims to provide a secure inference protocol with state-of-the-art inference time and accuracy in realistic networks with high latency. To do this, we take a co-design approach to balance accuracy and fast inference time. In Section 2.4, we develop MPC protocols that achieve the fastest evaluation of polynomials to date, assuming a modified ML architecture. In Section 2.5, we tailor the ML training procedure to achieve high accuracy using this modified architecture.

2.4 Faster Evaluation of Polynomials

In this section, we evaluate the speed-up of replacing ReLU’s with a naive polynomial approximation. We then develop our single-round protocols and show that they drastically reduce the activation function evaluation time in wide area networks.

2.4.1 The Polynomial Advantage

To highlight the speed-up of polynomials over standard ReLUs, we first evaluate the runtime of a single layer with 2^{15} ReLU activation functions in Figure 2.2. (See Section 2.6 for more implementation details.) First, we plot an unmodified version of CrypTen (using CryptGPU [190]) with the conversion to binary shares. Next, we replace the ReLU with a degree four polynomial fitted using least squares polynomial regression (see Section 2.5

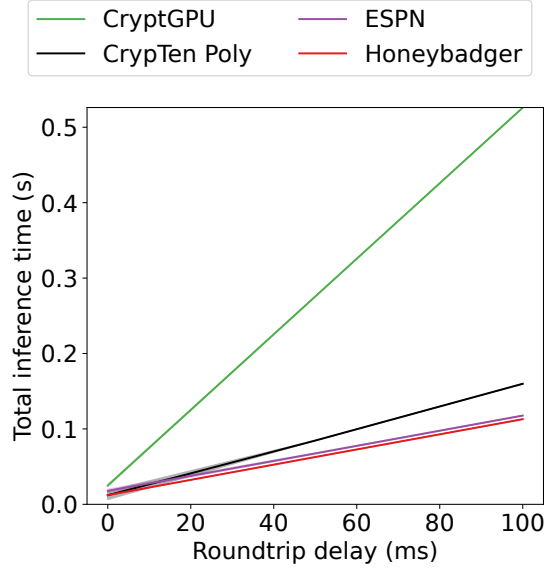


Figure 2.2: Benchmarking the secure evaluation of ReLU activation functions using various approaches. The x -axis is the network delay in ms and the y -axis is the mean runtime in seconds averaged over 20 runs with the shaded area representing the 95% confidence intervals.

for the details). We can see that using polynomials in off-the-shelf CrypTen is much faster across all network speeds than the default mixed arithmetic and binary protocol. This difference becomes more pronounced as we add more network delay or scale to deeper models with more ReLUs.

Despite the significant speed-up, naively computing a polynomial is still expensive in MPC with a non-trivial number of communication rounds. For example, Horner’s method (an iterative approach to evaluating polynomials) uses $O(n)$ communication rounds (where n is the degree of the polynomial). However, most MPC libraries (including CrypTen) use the square-and-multiply algorithm for exponentiation, followed by multiplying and summing the coefficients locally. The square-and-multiply algorithm requires $O(\log(n))$ multiplications (and thus rounds) in MPC. In practice, the default square and multiply implementation in CrypTen uses two rounds per ReLU for a degree four polynomial. To increase the advantage of using polynomial activation functions even further, we develop a new single-round protocol for evaluating polynomials in MPC.

2.4.2 ESPN: Exponentiating Secret Shared Values using Pascal's triangle

We present our single-round, highly parallelizable protocol ESPN for computing high-degree polynomials. The fundamental idea is utilizing the binomial theorem (Pascal's triangle) to achieve faster exponentiation. We begin by describing our protocol for raising a number $[[x]]$ to the power k , in MPC (see Algorithm 1 for an overview). Using the additive secret sharing scheme, the exponentiation corresponds to $(x_A + x_B)^k$ where x_A represents the first party's share and x_B represents the second (such that $x_A + x_B = x$). The binomial theorem expands this expression as:

$$x^k = (x_A + x_B)^k = \sum_{i=0}^k \binom{k}{i} x_A^{k-i} x_B^i \quad (2.2)$$

We observe that, for each i in the sum, party A can compute $\mathbf{a}_i = x_A^{k-i}$ without needing to communicate with party B (Alg. 1 line 4). Similarly, party B can compute x_B^i without communicating with party A (Alg. 1 line 5). Finally, $\binom{k}{i}$ can be computed by any party (or pre-computed ahead of time). For simplicity, we assign the computation of $\binom{k}{i}$ to party B . Thus party B computes $\mathbf{b}_i = \binom{k}{i} x_B^i$.

Once each party has computed their respective vectors, we multiply $\mathbf{a}_i \cdot \mathbf{b}_i$ for each i in parallel (Alg. 1 line 6). We carry out this multiplication using standard MPC protocols in one round. To use these multiplication protocols, each party must have a share of the input. We use a trivial additive secret sharing, where the other party inputs zero as their share to the protocol (Alg. 1 line 2). Finally, after the multiplication, the sum of the binomial theorem can be efficiently computed with no communication (Alg. 1 line 7).

Algorithm 1 Exponentiation Protocol for 2-party additive secret-sharing

```

1: procedure Exp( $[[x]]$ ,  $k$ )
2:    $\mathbf{a} = [[\mathbf{0}^k]]$ ,  $\mathbf{b} = [[\mathbf{0}^k]]$  ▷ Initialize shares
3:   for  $i = 0 : k$  do
4:     Party  $A$  computes:  $\mathbf{a}_i = \mathbf{a}_i + x_A^{k-i}$ 
5:     Party  $B$  computes:  $\mathbf{b}_i = \mathbf{b}_i + \binom{k}{i} x_B^i$ 
6:    $\mathbf{p} = \text{BeaverMultiply}(\mathbf{a}, \mathbf{b})$  ▷ Parallel Multiplication
7:    $s = \sum_{i=0}^k \mathbf{p}_i$  ▷  $s = \mathbf{a} \cdot \mathbf{b}$ 
8:   return  $s$  ▷ secret-shares of  $x^k$ 

```

2.4.3 Polynomial Evaluation with ESPN

Floating Point Considerations. Our initial description of ESPN considers the integer domain for simplicity. Extending to floating point values is straightforward but requires rescaling (a standard practice in fixed-point arithmetic). We use CrypTen’s two-party, local truncation protocol to ensure we do not incur additional rounds. However, there is a negligible chance of an incorrect result from this truncation protocol due to wrap-around in the ring. Specifically, the probability of an incorrect result when truncating x is $\frac{|x|}{2^L}$ where 2^L is the size of the ring [121, Appendix C.1.1]. This implies that x must be small compared to the ring for this fast truncation protocol to be correct.

Polynomial Evaluation Protocol. Using our exponentiation protocol, we show how to compute high-degree polynomials efficiently in a single round. We overview the protocol in Algorithm 2. First, in parallel, we compute all needed exponents using Algorithm 1. We recall that to ensure the correctness of truncation, we must ensure all intermediate values remain small. For simplicity, we define small to be that no intermediate scale becomes larger than twice the working precision p . We show the complete failure probability calculations of Algorithm 2 in Section 2.4.4.

To ensure the values remain small after exponentiation, we create multiple scaled-down copies of the input x , proportional to each exponent we need to calculate. To compute x^i we first scale x down by $2^{-\bar{s}}$ where $\bar{s} = \lceil (i-2)p/i \rceil$ and p is the current working precision of x (line 5). $\bar{s}$ is chosen such that 2^p becomes approximately $2^{2p/i}$ after scaling and thus approximately 2^{2p} after exponentiation by i . These are approximations as not all values of i divide $(i-2)p$, so there is some error from taking the ceiling. To account for the additional factor of approximately two, we do an additional rescaling after each exponentiation in line 7. This rescaling incorporates $\bar{s}$ (which includes the ceiling function) to ensure all values are scaled back to 2^p . Finally, after computing all powers, we can locally multiply the result by the coefficients (public values) and sum in line 8. We give a complete proof of correctness for Algorithm 1 and 2, including the truncation operator in Section 2.4.4.

Hyperparameter Restrictions. While Algorithm 2 is designed to keep intermediate values small, multiple hyperparameters determine the protocol’s effectiveness. Using results from Section 2.4.4 (namely the max of Theorem 2.4.2 and Theorem 2.4.3), we get that the probability of failure for a given truncation is bounded by

$$\Pr[\text{Truncation Failure}] \leq \frac{2^{n(\lceil \log_2 \lambda \rceil + 1) + 2p}}{2^L}. \quad (2.3)$$

Algorithm 2 Polynomial Evaluation Protocol for 2-party additive secret-sharing

```

1: procedure  $Poly([x], \alpha, n)$ 
2:    $\mathbf{p}_1 = x$ 
3:   for  $i = 2 : n$  do                                      $\triangleright$  In parallel
4:     Let  $\bar{s} = \lceil (i-2)p/i \rceil$                                 $\triangleright$  Scale down factor
5:      $x'_i = x \cdot 2^{-\bar{s}}$                                     $\triangleright$  Scale down before Exp
6:      $\mathbf{p}_i = \text{Exp}(x'_i, i)$                                 $\triangleright$  From Algorithm 1
7:      $\mathbf{p}'_i = \mathbf{p}_i * 2^{-p*(i-1)+\bar{s} \cdot i}$               $\triangleright$  Scale down after Exp
8:    $y = \alpha_0 + \sum_{i=1}^n \alpha_i \cdot p'_i$                   $\triangleright$  Locally dot product
9:   return  $y$                                                  $\triangleright$  Secret-shares of  $f(x)$ 

```

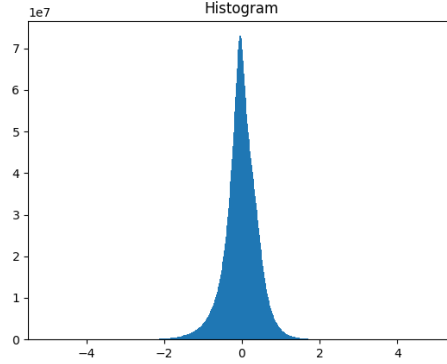


Figure 2.3: Polynomial Input Distribution of ResNet18 on CIFAR10.

For our experiments, we use CrypTen, with $L = 64$ [121] and a default working precision of $p = 16$. Assuming default values of $n = 4$ and $\lambda = 5$ we get a failure probability bound of 2^{-16} . However, this is a pessimistic upper bound since we consider the worst-case input of ± 5 . Conversely, in our experiments, we find that the distribution of inputs follows an approximately Laplace distribution. We plot the histogram of the input to all activation functions of a ResNet 18 model on CIFAR-10 in Figure 2.3. Thus, we observe a much smaller empirical failure probability.

While our default parameters were experimentally chosen to give high classification accuracy (Section 2.6, shows a minor degradation over plaintext accuracy), it is unclear if these values are optimal. For example, one approach to reducing the failure probability is to decrease p from 16 to 10-bit (which gives a failure probability of 2^{-28}). However, the trade-off is that the intermediate x'_i values will be truncated severely. For instance, with $n = 4$, x'_i

will be truncated to 5-bits. We find that this loss of precision is too significant to simulate a ReLU function accurately. Similarly, while a higher degree polynomial might better approximate a ReLU, a higher degree will negatively affect both the failure probability and the precision loss of x'_i . Future work could conduct an extensive hyperparameter search of all parameters to find the optimal trade-off. However, as we see in Section 2.6 (Tables 2.3-2.6), this would *at most* yield a 0.5% increase in encrypted classification accuracy (PyTorch vs. CrypTen accuracy), for the worst model and dataset.

Evaluating Algorithm 2. In Figure 2.2, we plot this approach alongside the previous approaches to evaluate the runtime. ESPN incurs slightly more overhead in the LAN setting; however, it scales significantly better (the confidence intervals do not overlap) to wide area networks that can be expected in practice.

2.4.4 Correctness Proofs

We begin by proving the correctness of Algorithm 1.

Theorem 2.4.1. *Given an input $[[x]] = x_A + x_B$ and exponent k , Algorithm 1, correctly returns $[[x^k]]$.*

Proof. We begin with party A . In line 4, they compute their share of the vector $\mathbf{a}$ where $\mathbf{a}_i = x_A^{k-i}$ (since $\mathbf{a}$ was initialized to zero and the for loop iterates over each entry of $\mathbf{a}$ exactly once). Similarly, in line 5, party B computes their share of $\mathbf{b}$ where $\mathbf{b}_i = \binom{k}{i} x_B^i$. We recall that party B 's share of $\mathbf{a}$ is the zero vector, and similarly for party A 's share of $\mathbf{b}$. Then, the vector $\mathbf{p}$ is obtained by multiplying $\mathbf{a}$ and $\mathbf{b}$ element wise in line 6. Thus, $\mathbf{p}_i = \mathbf{a}_i \cdot \mathbf{b}_i = x_A^{k-i} \binom{k}{i} x_B^i$. The final step (line 7), simply sums $\mathbf{p}$. Therefore,

$$s = \sum_{i=1}^k \mathbf{p}_i = \sum_{i=1}^k \binom{k}{i} x_A^{k-i} x_B^i \quad (2.4)$$

which applying the binomial theorem (2.2) gives $(x_A + x_B)^k = [[x^k]]$. □

Given the correctness of Algorithm 1, we now prove the correctness of Algorithm 2. To begin, we bound the failure probability of each truncation step in Algorithm 2.

Theorem 2.4.2. Consider computing a degree n polynomial fitted to the range $[-\lambda, \lambda)$. Let the global precision (size of the ring) be L -bit and the working precision of each value be p -bit. Then, the truncation in line 5 of Algorithm 2, fails (The local division of $[[x]_A, [x]_B] * 2^{-\bar{s}} \neq [[x * 2^{-\bar{s}}]]$) with probability at most

$$Pr[\text{Line 5 Failure}] \leq \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^L}. \quad (2.5)$$

Proof. Consider the first truncation of Algorithm 2 in line 5. The input to this truncation is x which we assume is contained in the range $[-\lambda, \lambda)$ with a working precision of p -bits. Therefore, the size of x is $2^{\lceil \log_2 \lambda \rceil + 1}$ in the integer part and 2^p in the decimal part. Which gives, $|x| \leq 2^{\lceil \log_2 \lambda \rceil + 1 + p}$. Given that we are working in a L -bit ring and the probability of failure of the truncation protocol is bounded by $|x|/Q$ where Q is the ring size [121], the result follows. $\square$

Theorem 2.4.3. Consider computing a degree n polynomial fitted to the range $[-\lambda, \lambda)$. Let the global precision (size of the ring) be L -bit and the working precision of each value be p -bit. Then, the truncation in line 7 of Algorithm 2 fails (The local division of $[[x]_A, [x]_B] * 2^{-p*(i-1)+\bar{s}*i} \neq [[x * 2^{-p*(i-1)+\bar{s}*i}]]$) with probability at most

$$Pr[\text{Line 7 Failure}] \leq \frac{2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p}}{2^L}. \quad (2.6)$$

where i is the power of x being truncated ($i \leq n$).

Proof. Consider the truncation in line 7 of Algorithm 2. The input to this truncation is the output of the previous truncation in line 5, raised to the power i . We assume the previous truncation was correct. Then after the truncation,

$$|x'_i| = \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^{\lceil (i-2)p/i \rceil}} \leq \frac{2^{\lceil \log_2 \lambda \rceil + 1 + p}}{2^{(i-2)p/i}} \quad (2.7)$$

for $i \in \{2, \dots, n\}$, where the inequality holds because $(i-2)p/i$ is positive. After applying the exponentiation by i we get $|\mathbf{p}_i| \leq 2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p}$. Given that we are working in a L -bit ring and the probability of failure of the truncation protocol is bounded by $|x|/Q$ where Q is the ring size [121], the result follows. $\square$

Theorem 2.4.4. Given an input $[[x]] = x_A + x_B$ and polynomial coefficients α , Algorithm 2, correctly returns the polynomial evaluation $[[\sum_{i=0}^n \alpha_i \cdot x^i]]$ except with probability.

$$Pr[\text{Algorithm 2 Fails}] \leq \frac{\sum_{i=2}^n \left(2^{i(\lceil \log_2 \lambda \rceil + 1) + 2p} + 2^{\lceil \log_2 \lambda \rceil + 1 + p} \right)}{2^L}.$$

Proof. First, prove correctness assuming the truncation operators are correct (then we will account for the probability of failure). We begin by proving that $\mathbf{p}'_i = x^i$ for $i \in \{1, \dots, n\}$ after line 7. We note that x is actually $x \cdot 2^p$ due to the fixed point encoding. $\mathbf{p}_1$ follows trivially. For $i \in \{2, \dots, n\}$, we work backwards, from line 7 to line 4 expanding the definition of $\mathbf{p}'_i$

$$\mathbf{p}'_i = \mathbf{p}_i * 2^{-p*(i-1)+\bar{s} \cdot i} \quad (2.8)$$

$$= (x'_i)^i * 2^{-p*(i-1)+\bar{s} \cdot i} \quad (2.9)$$

$$= (x \cdot 2^p \cdot 2^{-\bar{s}})^i * 2^{-p*(i-1)+\bar{s} \cdot i} \quad (2.10)$$

$$= x^i \cdot 2^p \quad (2.11)$$

where the first lines, up to (2.10), come from the substitution of lines 7 through 5, respectively. The remaining step follows from basic algebra.

Finally, we bound the failure probability of the algorithm. The truncation steps in line 5 and line 7 each introduce a possibility of wrap around error. Each truncation is executed $n - 1$ times. Thus, applying the bounds derived in Theorem 2.4.2 and Theorem 2.4.3 for each i in the for loop, the total failure probability follows. $\square$

2.4.5 Alternative Single Round Protocol: HoneyBadger

Like ESPN, Lu et al. give a single round protocol for exponentiation in MPC [142]. Despite focusing on a completely different problem (anonymous communication), they provide an MPC protocol of independent interest for exponentiation, which we also utilize in our work. They take a very different approach to our work that yields different trade-offs. Instead of the binomial theorem, their work utilizes the following factoring rule

$$x^k - r^k = (x - r) \sum_{i=0}^{k-1} x^{k-i-1} r^i \quad (2.12)$$

where r is a random secret-shared number derived during pre-computation. We assume each party has a share of x and a share of r^i for $i \in \{1, \dots, k\}$ before beginning the protocol (instead of the more common Beaver triplets). The first step in the protocol is to compute and reveal $x - r$ (x blinded by r), which uses a single round. Once revealed, this value becomes a public constant C . After some algebraic manipulation of (2.12), Lu et al. obtain a recursive formula for $x^i r^j$ given below.

$$[[x^k r^j]] = [[r^{k+j}]] + C \sum_{i=0}^{k-1} [[x^{k-i-1} r^{i+j}]] \quad (2.13)$$

Using dynamic programming, the parties can then compute any power ($x^k r^0$) using only additions of previously computed terms and powers of r . To compute polynomials using this protocol, we simply swap the call to *Exp* in line 6 of Algorithm 2.

The advantage of Lu et al.’s protocol is that the communication is small (only the opening of $x - r$). The primary disadvantage is that the protocol requires a modified pre-computation phase, which is as difficult to pre-compute securely as the original problem (it is exponentiation). On the contrary, our binomial protocol uses standard Beaver triplets commonly found in MPC frameworks. There are well established protocols for efficiently computing these triplets, and the parties may already have them due to the popularity of Beaver’s trick. A more minor disadvantage of Lu et al.’s solution is that, while the protocol requires very little communication, it is not locally parallelizable as each dynamic programming step depends on the previous one. In contrast, our entire protocol can be executed in parallel.

We also consider the runtime of using HoneyBadger in Figure 2.2. We emphasize this is a runtime-only evaluation. Without our training algorithm in Section 2.5, none of the polynomial-based solutions can attain usable accuracy. We find that ESPN and HoneyBadger perform similarly in practice, with HoneyBadger gaining a slight advantage in very low network delay. Due to the pre-computation trade-offs, we will evaluate both approaches for the remainder of this work.

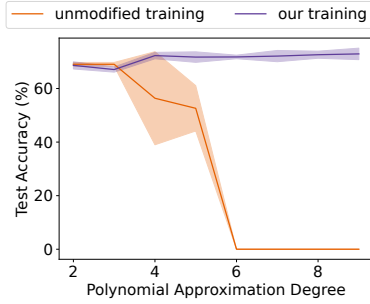


Figure 2.4: Accuracy of a 2-layer convolutional network trained with varying degrees for the polynomial activation function.

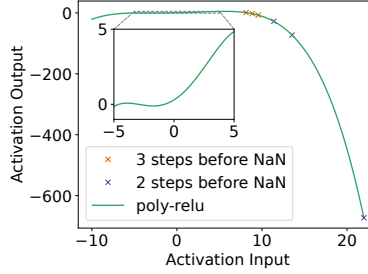


Figure 2.5: Illustrating the escaping activation problem for the two layers convolutional network.

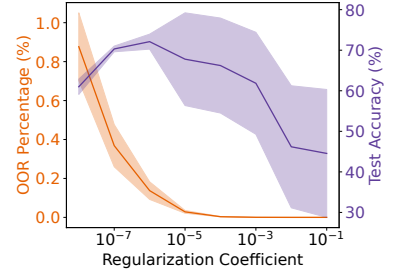


Figure 2.6: The effect of the regularization coefficient, β on model accuracy and out-of-range ratio for $\gamma = 10$

2.5 PILLAR: Polynomial Activation Regularization

Our initial benchmark in Section 2.4 showed a significant speed-up when replacing ReLU functions with polynomials implemented using ESPN and HoneyBadger. However, a notable challenge neglected thus far is that replacing a ReLU with a polynomial can drastically reduce the accuracy of the model [83, 107]. This section discusses the causes of the accuracy degradation and describes our mitigation techniques. Finally, we give empirical results showcasing the high accuracy of our modified training procedures across various architectures and datasets.

2.5.1 The Problem with Polynomial Activation Functions

Escaping Activations. The first step in replacing an activation function with a polynomial is to design a polynomial that closely approximates the original function. A common approach for this is the least-squares polynomial fitting. In this approach, a table of values is created for the polynomial over a small discretized range of values. This creates a system of equations for the polynomial coefficients that can be solved with least squares. The challenge with this approach is that, outside of this range, the polynomial no longer resembles the original activation function and often diverges rapidly. This leads to a problem called escaping activations, first identified by Garimella et al. [83]. If one naively swaps a ReLU for its polynomial approximation, all weights will become infinite within a few training epochs. We give an example of this degradation in Figure 2.4. We can see that, without modifying the training procedure, a polynomial can completely destroy the accuracy.

To illustrate the problem more clearly, we conduct an experiment using a polynomial of degree four fitted on the range $[-5, 5]$ ($\lambda=5$) as the activation function for a three-layer model on CIFAR-10 [123]. In Figure 2.5, we plot this polynomial activation function and the ℓ_∞ -norm of the input and output to each activation function. We note that, with no modification (except replacing ReLUs with polynomials), the weights of this model become undefined within approximately three epochs of training. First, we note the divergent behaviour of the polynomial outside the fitted range. Second, we observe the effect of the divergence on the outputs of the activation function. Specifically, we wait until the model weights become undefined (NaN in Python) and then observe the behaviour leading up to the explosion. We can see that three steps before the model weights become undefined (NaN), the input values of each activation are out-of-range, but the outputs still behave similarly to a ReLU. However, in the next iteration (two steps before NaN), a single value

in the first layer goes too far out of range. This causes a ripple effect for the other two layers, creating an extremely large output (approx -2000) in the final activation function. This large value creates a large gradient, and after another iteration of training, the values become so large that the gradients (and weights) become undefined (NaN). We find that minimizing the classification loss alone is not enough to keep the model in range as the gradients explode before decreasing the loss.

Truncated Polynomial Coefficients. An additional challenge is that we will evaluate the fitted polynomial in a finite ring with limited precision. This significantly impacts the polynomial coefficients, which tend to be relatively small, especially for the higher-order terms. Specifically, these small coefficients can get truncated to zero in limited precision, which causes the polynomial to diverge even inside the fitted range.

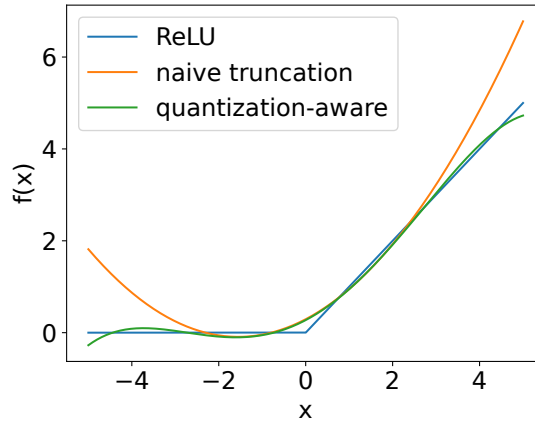


Figure 2.7: The effect of truncation on a polynomial activation function.

To show this, in Figure 2.7, we plot the polynomial approximation with and without our quantize-aware fitting approach. First, we plot the polynomial approximation after truncation and see that it diverges from a true ReLU. We also plot our quantized polynomial fitting and show that it addresses the problems of exploding activations within the range.

2.5.2 Defining PILLAR

Our approach, which we call PILLAR, is the combination of the components we describe in this section. Activation function regularization is our primary approach for mitigating

escaping activation functions. However, to scale to larger models, we find that the additional steps of clipping, regularization warm-up, and adding batch normalization are beneficial.

Quantization-Aware Polynomial Fitting. We begin by solving the problem of truncated polynomial coefficients. To address this, we fit the polynomial with the precision constraint in mind. We do this by using mixed integer non-linear programming. Let X be the set of all values between $[-\lambda, \lambda]$ in p -bit precision (the domain we want to fit on). First, we generate $Y = \text{ReLU}(X) \cdot 2^p$, a table of values for a standard ReLU scaled up by the precision. Scaling the output of the ReLU allows us to work in the integer domain (similar to fixed point arithmetic). We then compute a matrix B where each column is the different powers of X used in a polynomial ($B = [X^0, X^1, X^2, \dots]$).

Next, we solve the system $AB = Y$ for A using mixed integer linear programming with $A \in [-2^p - 1, 2^p - 1]$ to get the coefficients A that minimize the error between the polynomial AB and the ReLU values Y . Finally, we scale the resulting coefficients down by 2^p . We note that $A \in [-2^p - 1, 2^p - 1]$ corresponds to coefficients being bounded by $[-1, 1]$ after we scale down. We empirically choose $p = 10$ for all polynomials during the ML training. As we observe in Figure 2.7, our quantized polynomial fitting addresses the problems of exploding activations within the range. However, the issue of going out-of-range requires additional treatment.

Activation Regularization. Following the observations of Section 2.5.1 and Garimella et al. [83], it is clear that minimizing the classification loss alone is not sufficient to prevent escaping activations. Garimella et al. proposed QuaIL, a method that trains one layer of the model at a time, focusing not on classification accuracy but the similarity of the layer to a standard ReLU model [83]. QuaIL showed much better accuracy than naive training but only scaled to models with at most 11 layers.

In our work, we address the cause of the problem directly by regularizing the input to each activation function during training. We add an exponential penalty to the loss function when the model inputs out-of-range values to the polynomial activation function. Let x be the input to the activation function, and λ_{reg} be the upper bound of the symmetric range $[-\lambda_{reg}, \lambda_{reg}]$ in which we would like the input to be contained. Then, we define our penalty function as

$$p(x) = \left(\frac{x}{\lambda_{reg}} \right)^\gamma \quad (2.14)$$

where γ is a large even number (to handle negative values) determining the severity of the penalty. We find that values between six and ten work best in practice, with $\gamma = 10$ being

the default in our experiments. This penalty function gives negligible penalties (less than 1) for $|x| < \lambda_{reg}$ and rapidly grows (in the degree of γ) as $|x| > \lambda_{reg}$.

We aggregate $p(x)$ over $\mathcal{I}$, the set of inputs to all activation functions, by taking the average over each activation layer in the model. After aggregation, we scale the penalty using a regularization coefficient β and add it to the existing cross-entropy loss function of the model ℓ_c . Specifically, the modified loss function ℓ' is defined as:

$$\ell'(\cdot) = \ell_c(\cdot) + \frac{\beta}{K} \sum_{x \in \mathcal{I}} p(x) \quad (2.15)$$

where K is the number of activation layers in the model. This allows us to tune the importance of classification loss vs. the cost of going out-of-range.

Clipping. Although activation regularization teaches the model not to go out-of-range over time, the model still needs to avoid going to infinity during the early stages of training. Thus, during training, we apply a clipping function to the input of the activation function such that if any input goes out of range, it is truncated to the range’s maximum (or minimum) value. This clipping function does not affect the penalty as it is applied after the penalty function has been computed. We emphasize that this clipping function is only used during training and is removed during inference. The intuition is that the model should learn not to go out-of-range during training and thus no longer requires this clipping function during inference. Additionally, we find that setting the λ_{reg} of the penalty to be smaller than the range used for polynomial fitting (and clipping) can yield even better results. This is because the polynomial will be accurate for a larger range outside of the range the model was regularized to stay inside, allowing an extra buffer in case of failure during inference.

Regularization Warm-up. We find the minimum requirements for successfully training a model with polynomial activation functions are activation regularization and clipping. However, for larger models, the penalty term can be extremely large in the first few epochs (until the model learns to stay in range). In some cases, the loss can become infinite due to our regularization penalty. To address this challenge, we adopt a regularization scheduler for the first four epochs that slowly increases both γ and β to the values used for the rest of the training. Empirically, the following schedule works well and avoids infinite loss. We let $\gamma' \in \{4, 6, \dots, \gamma, \gamma, \dots\}$ and $\beta' \in \{\beta/100, \beta/50, \beta/10, \beta/5, \beta, \beta, \dots\}$.

BatchNorm Layers. Garimella et al. also investigated using normalization to help prevent the escaping activation functions [83]. They proposed a min-max normalization approach where each layer’s minimum and maximum values are approximated using a weighted moving average of the true minimum and maximum. These values are frozen during inference. Garimella et al. observed that this approach alone was insufficient, as activations still escaped the range during inference. We observe this operation is similar to the batch norm layer commonly added to ML models. The main difference is that the mean and standard deviation of the batch are used to normalize the layer instead of the minimum and maximum values. By fixing the approximation of the mean and standard deviation during inference (following CrypTen [121]), this operation is very efficient in MPC.

	Without BatchNorm	With BatchNorm
Normal Relu	88.77 ± 0.11	90.05 ± 0.16
PolyRelu	82.99 ± 0.42	87.37 ± 0.13

Table 2.1: Comparing the effect of BatchNorm on MiniONN model.

To study the effect of a batch norm layer on our training process, we train a standard ReLU model and a model with polynomial activations both with and without batch norm layers. We use the MiniONN architecture and give the results averaged over three random seeds with 95% confidence intervals in Table 2.1. We find that batch norm layers improve both models. However, the improvement due to using batch norm is significantly greater when using polynomial activation functions. This is an intuitive result as batch norm helps keep each layer’s output bounded and thus reduces the work of our regularization function.

Summary of PILLAR We refer to PILLAR as the combination of all components described in this section. We note that the clipping and regularization warm-up components are used (only during training) to enable regularization by preventing the model from going to infinity. The regularization component ensures that the model will stay in range during inference. All three of these components play a crucial role in the success of PILLAR, as removing any of them will result in a model with unacceptable accuracy (due to infinite weights or escaping activations). The batch norm is the only optional component, which we give a small ablation study over in Table 2.1.

Dataset	Model	Plain Accuracy $\pm$ CI	
		ReLU	PILLAR
Cifar10	MiniONN	91.2 ± 0.17	88.1 ± 0.26
	VGG 16	92.6 ± 0.16	90.8 ± 0.11
	ResNet18	94.7 ± 0.09	93.4 ± 0.14
	ResNet110	92.8 ± 0.27	91.4 ± 0.18
CIFAR-100	VGG 16	70.9 ± 0.17	66.3 ± 0.22
	ResNet32	68.4 ± 0.46	67.8 ± 0.32
	ResNet18	76.6 ± 0.07	74.9 ± 0.14
ImageNet	ResNet50	80.8	77.7

Table 2.2: Plain-text Accuracy of PILLAR (5 runs).

2.5.3 Measuring PILLAR’s Effectiveness

The Regularization Coefficient. To show the effect of our regularization and coefficient β , we conduct an experiment using the same three-layer model on CIFAR-10 from Section 2.5.1. In Figure 2.6, the left y -axis gives the out-of-range ratio (OOR), defined as the ratio of activation function inputs that were not within the interval $[-5, 5]$. The right y -axis is standard classification accuracy, and the x -axis varies the regularization coefficient β . We observe that when the coefficient, β , is small, the model goes out-of-range often and thus has poor accuracy. As we increase β , the out-of-range ratio decreases, and accuracy increases. However, if we increase the coefficient too much, the accuracy decreases again.

End-to-end Accuracies. We evaluate PILLAR across a range of different models and architectures considered in related work [107, 163, 190]. We summarize the results in Table 2.2. We include the accuracy of a model trained with standard ReLUs as a baseline. All results are averaged over five random seeds, and we show the 95% confidence interval. The only exception is ResNet50 on ImageNet, where we only train a single model due to the size of the dataset. We defer to Section 2.6 for the details of the experimental setup. We note that these results are using PyTorch with no cryptography or quantization. We give a complete evaluation using MPC in Section 2.6 where quantization has an effect. Table 2.2 provides preliminary evidence that our polynomial training approach yields high accuracies competitive with state-of-the-art ReLU models across a range of models and datasets.

2.6 Evaluation of Co-Design

In this section, we provide an end-to-end comparison of our co-design against state-of-the-art solutions in secure inference. We evaluate the performance of PILLAR and Algorithm 2 using both ESPN and HoneyBadger as they offer different trade-offs in the type of pre-computation needed and the size of the communication. We determine the state-of-the-art works following a recent SoK by Ng and Chow [164]. Specifically, we consider three solutions on the Pareto front of latency and accuracy as determined by Ng and Chow. These works are COINN [107], GForce [163] and CryptTen (CryptGPU) [121, 190].

We also evaluate Cheetah [101], a recent work not included in the SoK. We will evaluate the metrics of latency (or runtime of a single sample) and encrypted accuracy. We additionally evaluate the communication and number of rounds in Section 2.7. We begin with the experimental setup, then evaluate both metrics (runtime and accuracy) against each related work. Section 2.6.2 evaluates the ResNet-18 architecture, which gives our state-of-the-art performance. Section 2.6.3 evaluates the VGG-16 architecture, the only architecture GForce evaluates. Section 2.6.4 considers other ResNets and the MiniONN architecture following COINN. Finally, in Section 2.6.5, we evaluate ImageNet against Cheetah, COINN, and CryptGPU.

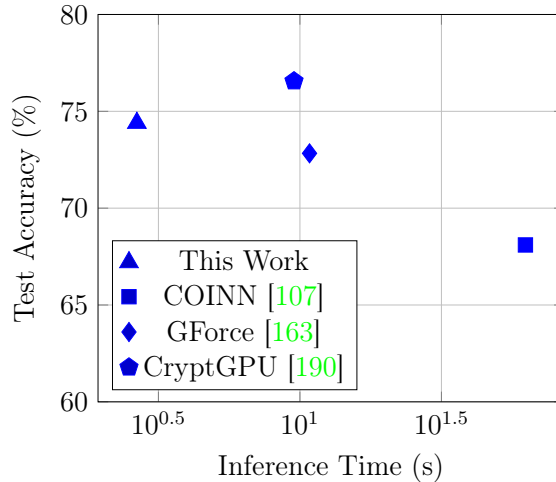


Figure 2.8: Summary of the inference time vs accuracy for each state-of-the-art approach on the CIFAR-100 dataset in the WAN (100 ms roundtrip delay).

Results Summary. We plot a summary of the accuracy and inference time for CIFAR-100 in Figure 2.8. For both datasets (recall Figure 2.1), we observe that our work always gives the solution with the fastest inference time by a statistically significant amount. In terms of accuracy, our work is competitive with the state-of-the-art, but CryptGPU is always the most accurate as it can infer unmodified plaintext models. Our solution is faster than CryptGPU by $4\times$, GForce by $5\times$, and COINN by $18\times$ on average in wide area networks. Our accuracies are competitive with state-of-the-art and plaintext solutions and stay stable (no escaping activations) with models containing up to 110 layers and 23 million parameters.

2.6.1 Experimental Setup

We develop an experimental setup that follows as closely as possible to the works we compare to [107, 163, 190]. We use CIFAR-10/100 [123] and ImageNet [53], the same common benchmark datasets as related work. Our model architectures include: MiniONN [137], VGG [183], and ResNets [97]. This covers models of depth 7 to 110 layers with the number of trainable parameters ranging from 0.2 to 23 million.

Implementation Details. All experiments are run on a machine with 32 CPU cores @ 3.7 GHz and 1 TB of RAM with two NVIDIA A100 with 80 GB of memory. We simulate network delay by calling the sleep function for the appropriate time whenever the client and server communicate. We simulate the LAN with 0.25 ms roundtrip delay and the WAN with 100 ms, following COINN [107]. We additionally evaluate a real WAN using AWS instances in Section 2.6.6. All experiments (except ImageNet) are repeated over multiple random seeds, and we report the mean and 95% confidence interval as shaded areas.

For all related work, we run our own benchmarks of their code unmodified. While results for ResNet32 and MiniONN appear in the Cheetah paper, there was no source code for these models so we only evaluate Cheetah on ImageNet. To use CryptGPU in practice, one must first train a model in PyTorch. For ImageNet, PyTorch provides pre-trained models. However, we will need to train a model for all other architectures and datasets. We simply use the same configurations as our PolyRelu models but with standard ReLUs. We include all source code to reproduce our results [1].

Hyperparameters. We introduce five new hyperparameters associated with our techniques: polynomial degree (n), polynomial approximation range (λ), polynomial regularization range (λ_{reg}), polynomial regularization coefficient (β), and polynomial regularization

exponent (γ). The default values for each are decided by extensive grid searches. These parameters primarily affect the accuracy and not the inference time, except for the polynomial degree (n), which has a minor effect on the communication size (but not the rounds). We found the optimal quantization-aware polynomial degree (n) to be 4. We found that higher degrees than four increase the failure probability of truncation with minimal accuracy gains (Section 2.4.3). Conversely, lower degrees give lower accuracy (as shown in Figure 2.4). The polynomial used in all evaluations is: $0.31445312 + 0.5x + 0.15625x^2 - 0.00292969x^4$.

For all models and datasets, we found a value of $\gamma = 10$ performs well as it introduces a strong enough incentive for PolyReLU inputs to stay in range (smaller values did not) while keeping penalization for values in range practically 0 (if an input x is within range, then $\frac{x}{\lambda_{reg}} < 1 \Rightarrow (\frac{x}{(\lambda_{reg})^{10}} \sim 0)$). Larger values than this often lead to an infinite penalty term. We found that a polynomial approximation range $\lambda = 5$ provides a good compromise between regularization and quantization. We found that smaller values of λ destroy the accuracy during training, and larger values use too much precision, increasing the failure probability. The optimal polynomial regularization range (λ_{reg}) and polynomial regularization coefficient (β) vary per model and dataset, although we found $\lambda_{reg} = 4.8$ (slightly tighter than $\lambda = 5$) and $\beta = 5 \times 10^{-5}$ to be good default values. We use $p = 10$ during ML training, but revert to CrypTen’s default precision of $p = 16$ during inference in a 64-bit ring.

We used Stochastic Gradient Decent as the optimizer with a learning rate of 0.013 as the default. This includes a Cosine Annealing Learning Rate Scheduler with Linear Learning Rate Warmup of 5 epochs and decay 0.01. We use a weight decay of either 10^{-4} or 5×10^{-4} and a momentum of 0.9. We used a default batch size of 128 and set the default number of Epochs to 185. For some models, we tuned the learning rate, number of epochs, and regularization coefficient to achieve a slightly higher accuracy. We detail hyperparameters in our source code repository.

2.6.2 ResNet-18 Architecture

In this section, we use a ResNet-18 architecture as it is the architecture that yields the best inference time and accuracy over all CIFAR-10 and CIFAR-100 experiments. For this comparison we focus on CryptGPU, which has been shown to be a state-of-the-art solution [164]. CryptGPU (or CrypTen) [190] serves as a baseline in all our comparisons including those against GForce and COINN in Sections 2.6.3 and 2.6.4. Neither COINN nor GForce support the ResNet-18 architecture evaluated in this section.

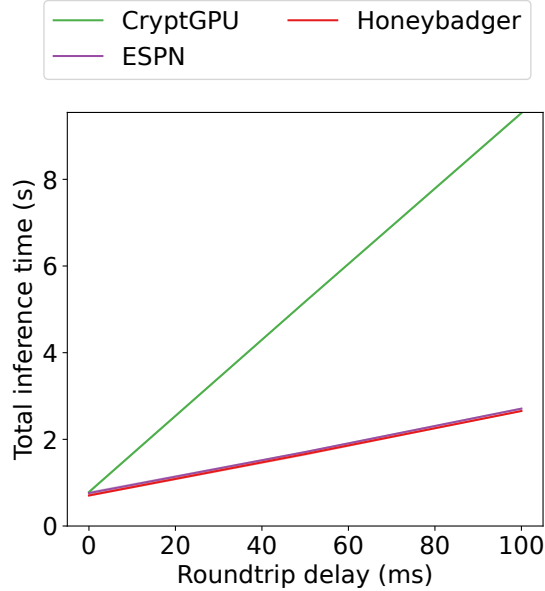


Figure 2.9: ResNet-18 evaluation on CIFAR-100 (20 runs).

Inference Time. We measure the inference time of a single input image over varying network delays. The results are given in Figure 2.9. We include the result for CIFAR-100 and omit the plot for CIFAR-10 as it displays similar trends. We observe that both PILLAR + HoneyBadger and PILLAR + ESPN outperform CryptGPU with statistical significance across all roundtrip delays (as the shaded area does not overlap). In the WAN (100 ms), this corresponds to a $4\times$ speedup over CryptGPU. Furthermore, we find HoneyBadger and ESPN perform similarly as observed in Section 2.4, with PILLAR + HoneyBadger having a slight advantage.

Accuracy. We measure the accuracy of the models on the testing set both in plain (using PyTorch) and encrypted (using CrypTen). We give the result in Table 2.3. First, we observe the plain and encrypted accuracies are very similar, indicating that quantization has a minor effect despite not considering this in training. We find that CryptGPU and PILLAR give similar accuracies, with CryptGPU performing slightly better, as is to be expected since they use unmodified activation functions. However, we argue this slight loss in accuracy is well justified by the significant decrease in inference time.

Dataset	Technique	Plain Acc	Enc Acc
CIFAR-10	PILLAR	93.4 ± 0.14	93.3 ± 0.22
	CryptGPU	94.7 ± 0.09	94.6 ± 0.10
CIFAR-100	PILLAR	74.9 ± 0.14	74.4 ± 0.31
	CryptGPU	76.6 ± 0.07	76.6 ± 0.13

Table 2.3: ResNet-18 accuracy comparison (5 runs).

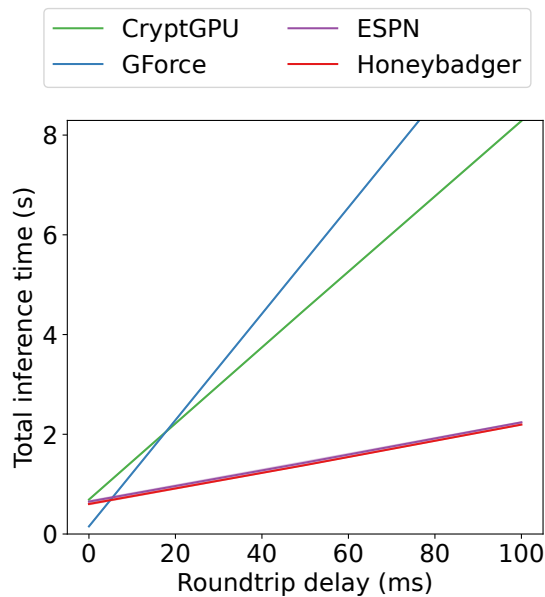


Figure 2.10: VGG-16 evaluation on CIFAR-100 (20 runs).

2.6.3 VGG-16 Architecture

In this section, we compare with GForce, the current state-of-the-art as shown by Ng and Chow [164]. GForce focused on a modified VGG-16 [183] architecture and compared it to all other works (including those using different architectures). For completeness, we evaluate the VGG-16 architecture using our techniques, and CryptGPU although we note that the ResNet-18 architecture outperforms VGG-16 in both inference time and accuracy. COINN [107] does not give results for VGG-16, so we exclude it from this section.

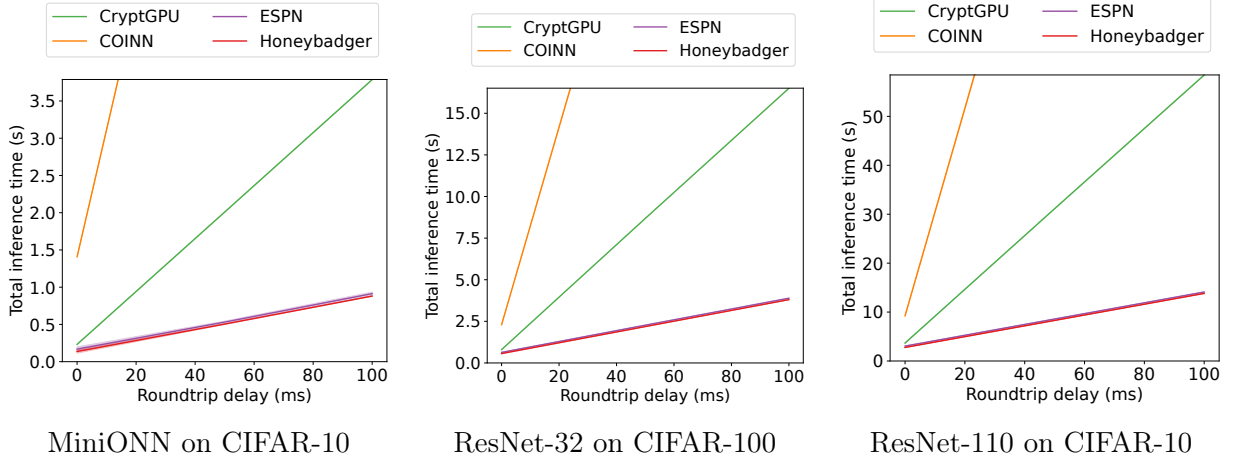


Figure 2.11: Evaluating the various COINN architectures (20 runs).

Inference Time. Since our work aims to reduce the rounds needed by binary non-linear layers, we replace the MaxPool layers in the VGG-16 with AvgPool for all solutions (including CryptGPU and GForce). We give the inference times over various delays in Figure 2.10. First, we note that GForce significantly outperforms all other solutions in the LAN. However, for more realistic high latency networks ($>5\text{ms}$ roundtrip delay), we observe our solutions significantly outperform GForce ($5\times$ speedup in WAN). Once again, our solutions outperform CryptGPU for all network delays.

Encrypted Accuracy. We recall that we swap the MaxPool layers for AvgPool layers in the inference time evaluation. This comes at a cost to accuracy for the VGG architecture. Thus, to give the best scenario possible for GForce, we consider the accuracy of GForce with MaxPools and our work with AvgPool. We give the results in Table 2.4. As expected, GForce outperforms our work in accuracy (due to the MaxPools); however, only by a few percentage points. We train CryptGPU to use AvgPool and find it also loses a few percentage points, confirming our hypothesis that a MaxPool is necessary for high accuracy in VGG-16. We emphasize that our ResNet-18 result outperforms GForce’s VGG result in inference time and accuracy. Furthermore, ResNets are a more popular and compact architecture due to skip-connections.

Dataset	Technique	Plain Acc	Enc Acc
CIFAR-10	PILLAR	90.8 ± 0.11	90.8 ± 0.14
	CryptGPU	92.6 ± 0.16	92.5 ± 0.16
	GForce	-	93.12
CIFAR-100	PILLAR	66.3 ± 0.22	66.3 ± 0.32
	CryptGPU	70.9 ± 0.17	70.8 ± 0.13
	GForce	-	72.83

Table 2.4: VGG-16 accuracy comparison (5 runs).

2.6.4 Other Architectures

While GForce is the current state-of-the-art, COINN is a competitive solution that evaluates ResNet architectures. Thus, we also evaluate the same configurations as COINN. This includes the smaller MiniONN architecture, a ResNet-32, and a ResNet-110. We exclude GForce from this evaluation as they only evaluate VGG-16 models.

Inference Time. We again swap all MaxPool layers for AvgPool in our work, and CryptGPU but leave COINN unmodified. We give the results in Figure 2.11. We observe that, over each of the three increasingly large architectures, the trends are similar and proportional to the number of parameters (0.2, 0.5, and 1.7 million parameters for MiniONN, ResNet-32, and ResNet-110, respectively). Across all architectures and network delays, our work outperforms COINN by a statistically significant amount ($18\times$ on average in WAN). We once again outperform CryptGPU in all evaluations¹ with a $4\times$ speed up on average in the WAN.

Encrypted Accuracy. We give the results in Table 2.5. We observe that PILLAR is competitive with related work in all models, although we remark that, once again, our ResNet-18 models outperform all others. We also note that, while COINN does quantization-aware training, PILLAR does not and still only loses a small amount of accuracy in encryption vs. plaintext.

¹All of which are statistically significant except MiniONN in LAN where the confidence intervals overlap slightly.

Dataset/Model	Technique	Plain Acc	Enc Acc
CIFAR-10 / MiniONN	PILLAR	88.1 ± 0.26	87.9 ± 0.46
	CryptGPU	91.2 ± 0.17	91.2 ± 0.16
	COINN	-	87.6
CIFAR-10 / ResNet-110	PILLAR	91.4 ± 0.18	91.4 ± 0.25
	CryptGPU	92.8 ± 0.27	92.7 ± 0.26
	COINN	-	93.4
CIFAR-100 / ResNet-32	PILLAR	67.8 ± 0.32	67.8 ± 0.47
	CryptGPU	68.4 ± 0.46	68.5 ± 0.45
	COINN	-	68.1

Table 2.5: Accuracy comparison on the various architectures considered in COINN (5 runs).

2.6.5 Scaling to ImageNet

In this section, we evaluate the scalability of our approach on the ImageNet dataset using a ResNet-50 architecture with 23 million parameters. This architecture was previously too large for training with polynomial activation functions [83]. We compare our approach to Cheetah, COINN and CryptGPU and exclude GForce as they do not consider ImageNet.

Inference Time. We plot the inference time in Figure 2.12. We observe a significant reduction over Cheetah and COINN across all network delays. We outperform Cheetah by $39\times$ in the LAN (0.25 ms) and $15\times$ in the WAN (100 ms). Over COINN, we observe a $28\times$ reduction in the LAN and a $90\times$ reduction in the WAN. Compared to CryptGPU we find that PILLAR + HoneyBadger is the fastest in all network delays by $3\times$ on average. PILLAR + ESPN is slightly slower in the LAN, but once again outperforms CryptGPU in the WAN.

Encrypted Accuracy. We present a summary of the accuracies in Table 2.6. We note that Cheetah [101] did not measure accuracy in either their code or paper so we omit them from this comparison. We observe a much higher encrypted accuracy for PILLAR compared to COINN and thus, our solution is Pareto dominant. For CryptGPU, we use a pre-trained PyTorch model with state-of-the-art accuracy. Therefore, as expected, CryptGPU has an accuracy 3% higher than the model we trained from scratch. We note that with a higher

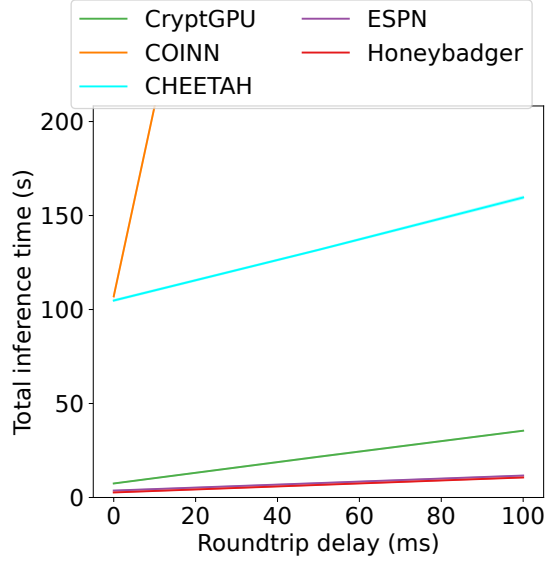


Figure 2.12: ImageNet evaluation on ResNet-50 (20 runs).

degree polynomial, we were able to train a 79.2% polynomial model. However, this model is not possible to infer in the 64-bit field used by CrypTen (as higher degrees need more precision by equation 2.3). We discuss future directions to further improve this result in Section 2.8.

Technique	Plain Acc	Enc Acc
PILLAR	77.7	77.3
CryptGPU	80.8	80.8
COINN	-	73.9

Table 2.6: ImageNet accuracy comparison (1 run).

2.6.6 Real WAN Evaluation

This section gives benchmarks for ESPN, HoneyBadger, and CryptGPU [190] in a real WAN. We use two AWS EC2 `g4dn.metal` instances, one in the Ohio data centre and one in Frankfurt, Germany. Each machine has 96 cores, 384 GB of memory, 100 GB/s network bandwidth, and a NVIDIA T4 GPU. In practice, we measured 10.8MB/s bandwidth between the two instances. We run all the ResNet architectures and summarize the

Dataset / Model	ESPN	HoneyBadger	CryptGPU
CIFAR-10 / ResNet-110	49.3 ± 0.1	49.0 ± 0.1	242.5 ± 0.9
CIFAR-10 / ResNet-18	15.3 ± 0.2	12.9 ± 0.1	48.9 ± 0.1
CIFAR-100 / ResNet-18	15.4 ± 0.2	12.9 ± 0.1	48.9 ± 0.5
CIFAR-100 / ResNet-32	14.1 ± 0.1	14.0 ± 0.1	75.4 ± 1.6
ImageNet / ResNet-50	153.9 ± 1.0	104.9 ± 0.8	268.67 ± 1.3

Table 2.7: Real World WAN evaluation. We report total inference time in seconds.

results in Table 2.7. We repeat each experiment 20 times and report the mean and 95% confidence intervals. We observe similar trends to the simulated WAN used in previous experiments. Namely, ESPN and HoneyBadger perform similarly in runtime for most models. The exception is larger models like ResNet50, where the bandwidth limits impact ESPN more than HoneyBadger. In all cases, CryptGPU is significantly outperformed by both approaches.

2.7 Communication and Rounds Benchmark

Dataset	Model	ESPN	HoneyBadger	CryptGPU	COINN	GForce	Cheetah
CIFAR-10	ResNet-18	0.46 (38)	0.23 (38)	0.45 (174)	/	/	/
	ResNet-110	0.55 (221)	0.13 (221)	0.53 (1093)	6.8	/	/
	VGG-16	0.32 (31)	0.26 (31)	/	/	0.050	/
CIFAR-100	ResNet-18	0.46 (38)	0.23 (38)	0.45 (174)	/	/	/
	ResNet-32	0.16 (65)	0.037 (65)	0.15 (313)	1.9	/	/
	VGG-16	0.32 (31)	0.26 (31)	/	/	0.050	/
ImageNet	ResNet-50	7.85 (160)	4.04 (160)	7.70 (552)	122.0	/	2.36 (1042)

Table 2.8: Evaluation of communication in GB and the number of rounds (shown in parenthesis).

We study two additional evaluation metrics of rounds and communication in this section. Recall that the number of rounds significantly affects the protocol latency over WAN. The communication affects each round’s throughput, depending on the network

bandwidth. In Table 2.8, we summarize the communication in GB and the number of rounds (shown in parenthesis) for our work and related work. We note that both COINN and GForce do not log the communication or rounds in their code base. Thus, we report their communication numbers from the corresponding tables in the papers (COINN [107, Table 3] and GForce [163, Table 7]). Neither work evaluates the number of rounds.

In all cases, ESPN and HoneyBadger significantly dominate all evaluated related work in the number of rounds with a $3 - 5\times$ improvement. The communication of ESPN and HoneyBadger is significantly less than COINN, approximately the same as CryptGPU, and more than GForce and Cheetah. Our HoneyBadger solution has approximately $2\times$ the communication of Cheetah, but Cheetah has $10\times$ the rounds; thus, in practice, HoneyBadger gives much better performance as shown in Section 2.6. GForce has an impressively low communication online (50MB) due to offloading 20GB of communication to an offline phase. Once again, despite having higher communication, we recall that both ESPN and HoneyBadger dominate related work in runtime in the WAN (as shown in Section 2.6).

2.8 Discussion

Our experimental evaluation in Section 2.6 showed our algorithms significantly outperform all related work in WAN inference time. While state-of-the-art compared to other polynomial training approaches, PILLAR still incurs a minor accuracy degradation compared to standard models with ReLUs. We posit a few directions for future work to further close this gap between polynomials and ReLUs.

Quantization. Note that aside from our quantization-aware polynomial fitting described in Section 2.5, we have made no efforts to reduce the effects of quantization. COINN developed training algorithms to help the model be robust to the overflow and quantization present in MPC [107]. An interesting future work would be to combine the COINN methods with PILLAR to see if further accuracy gains are possible.

Precision. By using CrypTen as our backend, we were limited to a 64-bit ring for cryptographic operations. As discussed in Section 2.4.3, this precision determines the degree and range of polynomials we can use (due to either the failure probability of truncation or severe truncation of intermediate values). Interesting future work is to increase this precision to enable higher-degree polynomials and study the performance-accuracy trade-off.

Our initial results on ImageNet show that we can train up to a degree eight polynomial without suffering escaping activations. However, we could not increase the ring size to study the effect of higher degrees on inference time.

MaxPools. We recall that a MaxPool layer requires comparisons and, thus, expensive conversions to binary shares (like ReLUs). Therefore, we replaced all MaxPools with AvgPool layers. However, in some architectures, such as VGG-16 [183], we found that swapping MaxPool for AvgPool degraded accuracy by up to 6%. Finding an efficient MaxPool alternative for architectures like VGG is important for future work. However, since the ResNet models give high accuracy using AvgPool layers we did not pursue this issue further.

2.9 Related Work

This work focuses on achieving state-of-the-art run time and accuracy in two-party secure inference. We measure this objective by evaluating against the current state-of-the-art as determined by a recent SoK by Ng and Chow [164]. Namely, we compare to COINN [107], GForce [163] and CrypTen [121, 190] in Section 2.6 as they represent the Pareto front according to Ng and Chow [164]. Another potential candidate on the Pareto front is Falcon [132], with low latency and accuracy [132]. We did not evaluate Falcon as the accuracy drop was too significant (over 10% [164]). Furthermore, GForce is shown to outperform Falcon in both latency and accuracy, and we outperform GForce [163]. For a complete list of other works not on the Pareto front, we defer to Ng and Chow’s work [164]. Notably, many works consider different threat models or use different approaches, such as homomorphic encryption. We leave extending our polynomial activation functions to these settings for future work. For the remainder of this section, we discuss works with a similar approach to ours that are not state-of-the-art or not evaluated by Ng and Chow [164].

Replacing or Reducing ReLU’s. It has been established that the non-linear functions such as ReLU are the bottleneck for secure computation [83, 85, 107, 152]. Several works initially focused on reducing the number of ReLU activations, optimizing for the best trade-off between accuracy and runtime [85, 113]. A faster approach is to replace all ReLU’s entirely using polynomial approximations [83]. In Section 2.5, we discussed the most recent work in this space, Sisyphus [83]. While making significant progress toward training models with polynomial activations, Sisyphus could not overcome the escaping

activation problem for models with more than 11 layers. Before Sisyphus, there were a handful of works on smaller models that typically focus on partial replacement (some ReLU’s remained) [88, 152, 154]. An interesting exception from Lee et al. used degree 29 polynomials in HE but suffered prohibitively high runtimes [128]. Our work is the first to make high-accuracy polynomial training feasible (without escaping activations) in deep neural networks.

A notable recent work is PolyKervNets [17]. Inspired by the computer vision literature, PolyKervNets remove the activation functions and instead exponentiate the output of each convolutional layer [17]. The problem with this approach is that, similar to polynomial activation functions, the exponents make the training unstable. Aremu and Nandakumar note that exploding gradients prevent their approach from scaling to ResNet models deeper than ResNet18 (using degree 2 polynomials). Furthermore, PolyKervNets only allow for a single fully connected layer which reduces the accuracy of the models. Conversely, PILLAR scales to deeper models such as ResNet110 and much higher degrees. Moreover, we achieve significantly better plaintext accuracy on ResNet-18 (93.4 vs 90.1 on CIFAR-10 and 74.9 vs 71.3 on CIFAR-100).

Polynomial Evaluation in MPC. Our work focuses on co-designing the activation functions with cryptography by using polynomials. However, the problem of computing polynomials in MPC is of independent interest and has also been studied in the literature. The state-of-the-art in this space is HoneyBadger, as discussed in Section 2.4. Other notable works include the initial inspiration for HoneyBadger from Damgård et al. [50]. This early approach conducts exponentiation by blinding and reconstructing the number to be exponentiated so the powers can be computed in plaintext [50]. Building off this idea, Polymath constructs a constant round protocol for evaluating polynomials focused on matrices [141]. However, HoneyBadger outperforms Polymath by reducing both the rounds and the number of reconstructions to one.

2.10 Conclusion

In this work, we co-designed the ML and MPC aspects of secure inference to remove the bottleneck of non-linear layers. PILLAR maintains a competitive inference accuracy while being significantly faster in wide area networks using novel single round MPC protocols (ESPN and HoneyBadger). Our state-of-the-art inference times motivate future work to further improve the ML accuracy of polynomial activations in DNNs.

Chapter 3

Selective MPC: Distributed Computation of Differentially Private Key-Value Statistics

This chapter is adapted from work that previously appeared at the ACM SIGSAC Conference on Computer and Communications Security (CCS) [102]. In this system, we consider a set of client devices with data in the form of key-value pairs, such as usage data. An organization (such as a service provider) wishes to aggregate the data in a privacy-preserving fashion. We assume the clients do not trust the organization to see the raw data, only a privatized version of it that adheres to differential privacy. Previous solutions use local differential privacy where noise is added by the clients before submitting the data. The bottleneck of prior work is utility, as local noise added by clients significantly degrades the accuracy of the final statistics, especially for this type of data. In this chapter, we propose a secure computation-based alternative that yields much better utility. When using secure computation, the challenge instead becomes performance, which we address through a careful co-design of secure computation and differential privacy.

3.1 Introduction

Key-value data naturally occurs in many internet applications, such as video ad and mobile app analytics where keys are identifiers and values are time or frequencies [204]. In these applications, a provider collects key-value data from a large number of users to compute

aggregate statistics which provide crucial insights. However, as individuals become more privacy-aware, they become increasingly unwilling to divulge their sensitive data to a central party. Consequently, recent work has investigated systems that simultaneously allow for the computation of useful statistics while respecting the privacy of individuals and their data. A popular method for achieving this goal is to use local differential privacy (LDP). Under this model, users take privacy into their own hands, so to speak, by perturbing their data before sending it to a central party to be processed. This technique provides a limit on the amount of information the central party can infer about each user’s data while allowing the central party to compute a meaningful aggregate result. These properties have made local differential privacy a preferential solution for many organizations such as Google [76], Apple [56], Mozilla [72], and Microsoft [58] to collect user statistics. For example, the Chrome Web browser uses local differential privacy to collect aggregate statistics about settings such as the homepage and default search engine to counter software that maliciously alters these settings [76]. As another example, Mozilla experimented with using local differential privacy techniques to collect browser telemetry data in a private manner [72].

Applying LDP to key-value data is more challenging than numerical or categorical data. Naively applying noise to the key implies a high loss in the accuracy of aggregate statistics, since it breaks the correlation between the key and the value. Both Ye et al. [204] and later Gu et al. [94] develop improved LDP mechanisms that reduce this accuracy loss. While improving over naive solutions, this line of work is still confined by the inherent accuracy limitations of the LDP model. Intuitively, this is because each user individually adds randomness, which compounds in the final aggregated result.

An approach for achieving both the accuracy of the central model in DP and the trust of the local model is to use multi-party computation (MPC). A popular type of MPC uses secret sharing, wherein the parties jointly compute a function without any one party learning the input. Applying MPC to key-value data is challenging because it may introduce a prohibitively large computational overhead to keep the inputs concealed. Specifically, simultaneously computing over the keys while keeping them secret is a complex issue. To compute and update aggregate statistics, we require an array which maintains the totals for each key. If the key is secret shared, and thus kept confidential from each party, then indexing each value into this array using MPC incurs a cost linear in the size of the array. This results in a total computation cost on the order of $|\mathcal{S}|n$, where $|\mathcal{S}|$ and n denote the total number of key-value pairs and the number of distinct keys, respectively. An alternative approach, using ORAM [148], has polylogarithmic cost; however, requires large constants and is typically applied to huge data set sizes. Hence, these approaches remain impractical. If the key is revealed to the parties or encrypted in a way that reveals the access pattern

(e.g., public-key searchable symmetric encryption [49]), the parties will learn the count of each keyword, which violates DP. To prevent this violation, one could pad the counts by adding dummy key-value pairs, restoring an approximate differentially private guarantee. This requires adding an abundance of dummy records in order to achieve any reasonable privacy guarantees. Thus, this approach incurs a large communication cost.

In this paper, we design a new method of MPC called *selective multi-party computation*. Selective MPC does not conceal keys, resulting in an efficient protocol which does not incur prohibitively high communication costs. In typical MPC solutions, all parties receive shares for all key-value pairs. In contrast, our design involves randomly selecting a subset of parties to receive shares for each key-value pair. Thus, each party only views shares for a random subset of the entire dataset. In other words, our approach simulates removing some data from the view of each party, capturing subtractive noise (similar to how adding dummies in previous approaches captured additive noise). We compose the two approaches (selecting a subset of parties and adding dummies) to emulate a two-sided distribution without removing any data. We obtain a new theoretical result, proving that one can achieve pure differential privacy from joining two different one-sided distributions.

Our approach substantially reduces the computation costs, since our construction reveals keys and access patterns while preserving DP guarantees. The computation complexity of the MPC is reduced by a factor of $O(|\mathcal{S}|)$ over the naive MPC approach, where $|\mathcal{S}|$ is the number of key-value pairs. By simulating both subtractive and additive noise, selective MPC requires a very low number of dummy records. We produce 10-20 times less dummies than the padding approach and achieve low ϵ in a pure-DP model ($\delta = 0$). Compared to the previous LDP mechanisms, we improve accuracy by a factor of $O(\sqrt{m})$ where m is the number of clients. We give a detailed comparison to other approaches in Section 3.8.

In summary, our contribution is a secure multi-party computation of differentially private frequency and mean estimations over key-value data with the following properties.

- Distributed trust and provable security guarantees.
- Pure differential privacy in the local trust model.
- High accuracy independent of the number of users.
- Scalability up to 10,000 keys with results in at most 20 seconds.

3.2 Preliminaries

3.2.1 Differential Privacy

Differential privacy (DP), introduced by Dwork et al. [69], is a common privacy notion that enables the calculation of aggregate statistics on users' data in a privacy-preserving manner. Intuitively, differential privacy states that a user's participation has a bounded effect on the final output. More formally, differential privacy can be defined as follows.

Definition 3.2.1 (Differential Privacy). *A randomized algorithm $M : \mathcal{X}^n \mapsto \mathcal{Y}$ is (ϵ, δ) -DP, if for any pair of neighbouring datasets $X, X' \in \mathcal{X}^n$, and for any $T \subseteq \mathcal{Y}$ we have*

$$\Pr[M(X) \in T] \leq e^\epsilon \Pr[M(X') \in T] + \delta. \quad (3.1)$$

If $\delta \neq 0$, then we say that the mechanism provides approximate differential privacy. When $\delta = 0$ it satisfies pure differential privacy. There are two main trust models in differential privacy. In the central model, we assume the existence of a trusted curator that gathers the users' data, calculates the noisy aggregate statistic, and publishes it. In the local model, users send noisy data to a (potentially untrusted) curator to calculate the aggregate statistic. Although this model eliminates the need to trust the aggregator, it comes at the cost of utility. This is because there is noise added to each data point on the order of the number of clients.

The most common approach to satisfying DP is to add random noise to the output of a function. To prove that this satisfies DP, the function must have a bounded output. We formalize this notion as the sensitivity of a function.

Definition 3.2.2 (Sensitivity). *Let $f : \mathcal{X}^n \mapsto \mathbb{R}^k$. If D is a distance metric between elements of $\mathbb{R}^k$ then the D -sensitivity of f is*

$$\Delta_D^{(f)} = \max_{(X, X')} D(f(X), f(X')), \quad (3.2)$$

where (X, X') are pairs of neighbouring datasets.

Common examples of the distance metric are the ℓ_1 -norm and ℓ_2 -norm. With the notion of sensitivity, we can now define the Laplace mechanism, which uses sensitivity based on the ℓ_1 -norm, which we denote $\Delta_1^{(f)}$ for simplicity.

Definition 3.2.3 (Laplace Mechanism). *Let $f : \mathcal{X}^n \mapsto \mathbb{R}^k$. The Laplace mechanism is defined as*

$$M(X) = f(X) + \text{Lap}(\Delta_1^{(f)}/\epsilon)^n, \quad (3.3)$$

where $\text{Lap}(\Delta_1^{(f)}/\epsilon)^n$ represents n i.i.d draws from the Laplace distribution with parameter $\Delta_1^{(f)}/\epsilon$.

It is well known that the Laplace mechanism satisfies ϵ -DP [70].

3.2.2 Secret Sharing

A secret sharing scheme [31, 179] enables a secret s to be split into n shares to be distributed among a set of n participants. The secret is split such that the n shares permit reconstruction of s , but no set of $n - 1$ or fewer shares yield any information about the secret.

For example, consider the additive secret sharing scheme. In this scheme, to split the secret s , we select $n - 1$ shares $s_1, \dots, s_{n-1}$ at random from the domain and let $s_n = s - \sum_{i=1}^{n-1} s_i$. Then all players can recover the secret s by computing $s = \sum_{i=1}^n s_i$. Note that each share is computed as a fixed linear function of the secret and random elements of a field, so participants can locally compute any linear function of shared values. We use $[x]$ to denote that the value of x is shared among participants. For example, given constants v_1, v_2, v_3 and shares of values $[x], [y]$, one can locally compute

$$v_1[x] + v_2[y] + v_3 = [v_1 \cdot x + v_2 \cdot y + v_3]$$

to obtain shares of the value $v_1 \cdot x + v_2 \cdot y + v_3$. This property of the additive secret sharing scheme enables the construction of more general multi-party computation protocols [48].

3.2.3 Secure Multi-party Computation

Secure multi-party computation (MPC) allows a set of participants $\{p_1, \dots, p_n\}$, where p_i holds private data d_i , to jointly compute a function $f(d_1, \dots, d_n)$, while keeping their input private and revealing only the output. We focus on a variant of MPC which performs operations over shares of the data [29].

In MPC based on secret sharing, participants distribute shares of their data among a set of computation nodes. The computation nodes perform a computation over the secret shared data to obtain shares of a function of the participants' data, $f(d_1, \dots, d_n)$. Most

MPC protocols represent the desired function as a circuit using basic operations over a finite field (addition and multiplication) and provide subprotocols for performing those basic operations [51, 147].

We can consider two different models for secure multi-party computation: the *semi-honest* (or passive) model and the *malicious* (or active) model. In the semi-honest model, introduced by Goldreich [90, §7.2.2], adversaries do not deviate from the protocol but may gather information to infer private information. In the malicious model [90, §7.2.3], adversaries may deviate from the protocol by altering messages or other means to infer private information.

3.3 Problem Statement

Our goal is to present a protocol that permits the efficient computation of differentially private statistics over key-value data while sacrificing minimal trust to any party. A naive solution might first divide the key-value pairs into a set of keys and a set of values and then apply existing methods for computing differentially private histograms to each set. However, as Ye et al. [204] argue, this does not maintain the correlation between keys and values. For this reason, computing meaningful differentially private statistics over key-value data is more challenging than histograms.

3.3.1 Notation

We have a set of m clients, each denoted by an index $i \in [m]$. Each client has a set of key-value pairs $\mathcal{S}_i$. We denote the j th key-value pair of client i to be $\langle k_{ij}, v_{ij} \rangle$, where each $k_{ij} \in \mathcal{K}$ and each $v_{ij} \in \mathcal{V}$ for some sets $\mathcal{K}, \mathcal{V}$. The multiset of key-value pairs owned by all m clients is denoted $\mathcal{S}$. For a value or algorithm, we adopt the convention of using a subscript k to restrict it to a key k . We summarize the notation in Table 3.1.

In this work, we focus on the following two statistics over key-value data. Let $\mathcal{A}_k = \{\langle k_{ij}, v_{ij} \rangle \in \mathcal{S} \mid k_{ij} = k\}$ be the multiset of key-value pairs corresponding to key k .

- **Frequency estimation.** The frequency of some key k , q_k , is defined as the number of instances of the key k in the set $\mathcal{S}$. That is, $q_k(\mathcal{S}) = |\mathcal{A}_k|$. We denote the vector of frequencies of all keys by $q(\mathcal{S})$.

Symbol	Description
$\mathcal{K}$	domain of the keys
$\mathcal{V}$	domain of the values
n	the size of the key space $\mathcal{K}$
m	the number of clients
$\mathcal{S}$	multiset of all key-value pairs
$\mathcal{S}_i$	set of key-value pairs owned by client i
R	bound on the values in $\mathcal{V}$
ℓ	the number of computation nodes
t	size of subset of computation nodes
λ	maximum # of keys held by a client
γ	minimum frequency of any key
q	vector of frequencies
μ	vector of means
r	parameter for geometric distribution
p	ratio of t/ℓ

Table 3.1: Notation

- **Mean estimation.** The mean of the key k , μ_k , is the mean of all values in $\mathcal{S}$ whose key is k . Thus, the mean is

$$\mu_k(\mathcal{S}) = \frac{\sum_{\mathcal{A}_k} v_{ij}}{q_k(\mathcal{S})}.$$

We denote the vector of means for all keys by $\mu(\mathcal{S})$.

3.3.2 Assumptions

We aim to compute statistics which satisfy the requirements of differential privacy, as described in Section 3.2.1. In the case of telemetry data, frequency and mean are very helpful statistics. For example, a browser vendor may want to calculate how many users have added known trackers to a blocklist [72]. We make the following assumptions about the data in the set $\mathcal{S}$.

- **Bounded values.** We assume that the domain of the values is bounded by some interval of size $2R$. That is, every value $v_{ij} \in [x - R, x + R]$, for some x . We assume this to bound the sensitivity of statistics on the values.

- **Distinct keys.** We assume that each key appears in $\mathcal{S}_i$ at most once. This is done for simplicity because in the case where a client has multiple values for a single key, they could simply aggregate their data for said key beforehand.
- **Maximum size of $\mathcal{S}_i$.** We assume that each user has at most λ key-value pairs. That is, $|\mathcal{S}_i| \leq \lambda$. This is assumed to tighten the analysis but is not restrictive because in the worst case $\lambda = n$.
- **Minimum frequency.** We assume that the frequency of each key is at least γ , for some $\gamma \geq 1$. This is to bound the sensitivity of our mean calculation. We require this term to obtain a data-independent bound on the sensitivity since both the frequency and mean are sensitive in our setting.

We consider user-level differential privacy where neighbouring datasets change by adding or removing a single user’s key-value pairs. Here, the users correspond to the clients which we discuss throughout the paper. We apply concepts from secure multi-party computation to remove the need for a single trusted server. In particular, the clients distribute their data between a small set of untrusted servers, which collectively compute statistics over the data. We call these servers the computation nodes, and we require that there are $\ell \geq 3$ computation nodes. We make the following assumptions about the entities involved in the computation.

- **Anonymous channels.** We assume that there exist anonymous channels between clients and computation nodes. Section 3.5.3 discusses how our protocol functions in the shuffle model, where there is an intermediary party between the clients and the computation nodes [30, 73]. Our assumption of anonymous channels is a relaxation of the shuffle model.
- **Trust model.** For simplicity, we begin by describing our protocol in the semi-honest model, i.e., we assume that the clients and the computation nodes follow the protocol. Any party may however attempt to infer other information from the data that they observe (semi-honest model). Section 3.7 explains how to account for malicious clients and malicious computation nodes who do not follow the protocol.
- **Secure channels.** We assume pairwise secure and authenticated communication channels between the computation nodes. We also assume secure, anonymous communication channels between the clients and computation nodes, in the absence of a shuffler.

The final computed statistics may be published to all parties. Given this output and the information within their view throughout the protocol, the computation nodes should only be able to infer information that is bounded by our differential privacy guarantees. In terms of formally proving security, we aim to satisfy the definition of indistinguishable computationally differential privacy (IND-CDP) by Mironov et al. [151]. We discuss the security model in more detail in Section 3.7.

3.3.3 System Architecture

As described in the previous subsection, we have a set of clients and a set of computation nodes. First, the set of clients will send their data to the computation nodes. This is called the *data collection phase*. Then, in a *computation phase*, the computation nodes will compute some statistic and publish the output. In our solutions, we introduce an optional dummy generator who acts similarly to clients and we may assume the existence of an anonymous channel of communication between clients and computation nodes. We discuss alternatives to both of these assumptions in Section 3.5.3. The overall flow of data in our system can be seen in Figure 3.1.

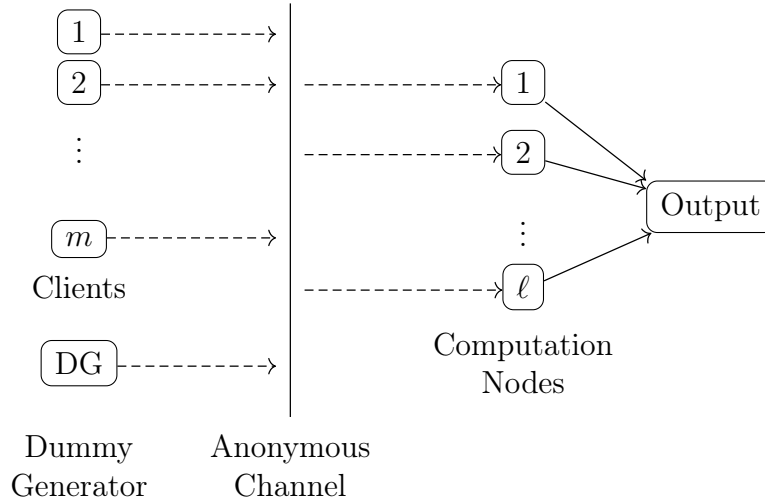


Figure 3.1: Data flow.

This architecture is similar to systems currently in use that involve a large set of clients sending data to multiple servers, such as Prio [9, 45]. The use of multiple servers for computation prevents the aggregation of private data at one central authority. Recall the

example use-case of private telemetry data collection. Clients in this setting represent the web browsers that send data to the servers responsible for the computation. For example, these servers may be controlled by browser vendors and/or certificate authorities [108].

3.4 Preliminary Solutions

In this section, we provide a high-level overview of some preliminary solutions that fit within the architecture and trust assumption. This ultimately helps us demonstrate how we came to our final solution.

We can distinguish between solutions which conceal keys and those which do not. Solutions that hide the keys incur a high computational cost to index the records into an array. So, as preliminary solutions, we only consider solutions that do not conceal keys. We begin with a naive solution that minimizes trust but requires high communication costs. The solution involves two types of participants: clients who own the data and computation nodes who calculate the statistic. We assume that there exists secure, anonymous channels between the clients and the computation nodes. We call this the *full padding* solution, and it is executed in the following two steps.

Step 1: Padding, Share Generation, and Encryption. Clients create a *dummy* key-value pair for each key not in their set, $\mathcal{S}_i$. For each key-value pair, the client applies a secret sharing scheme to generate ℓ shares of the value. The clients forward the key-share pairs to the corresponding computation nodes.

Step 2: Computation. Upon receiving the data, the computation nodes perform an MPC protocol to compute the differentially private statistic for each key and reconstruct the final result.

Since the computation nodes only see shares of the values, they cannot distinguish between real key-value pairs and dummies. Also, since every client sends a key-value pair, real or fake, for each key, it is impossible for the computation nodes to determine the true frequency of a key.

The obvious disadvantage to this protocol is the communication cost. Each client has n key-value pairs, for which they generate ℓ shares, resulting in a communication cost of $O(mn\ell)$. These costs are prohibitively expensive, so we consider methods of improving our approach. We remark that this cost is incurred to prevent the computation nodes from learning who has what data and the true frequency of the keys.

To improve upon this naive solution, we can leverage differentially private padding to mask the true frequency of the keys. This is accomplished by adding a semi-trusted dummy generator. The protocol, which we refer to as the *one-sided dummy* solution, requires the following three steps.

Step 1: Share Generation. For each key-value pair they own, the client applies a secret sharing scheme to the value and generates ℓ shares. They forward this data to the corresponding computation nodes.

Step 2: Padding. The dummy generator generates “enough” dummy key-value pairs to satisfy differential privacy. They generate shares for these new values via the same secret sharing scheme used by the clients. The data is forwarded to the corresponding computation node.

Step 3: Computation. The computation nodes receive all of the data and then perform a multi-party computation protocol to compute the desired differentially private statistic. They collectively output the result.

This solution employs differential privacy in two ways: to bound the information learned as a result of adding fewer dummies and to output a noisy statistic. In the former, we require a randomized algorithm to add dummies; however, only adding key-value pairs creates a one-sided distribution and thus has a chance of being blatantly non-private.

There exists work that uses a similar approach of only adding fake records from one-sided distributions in different settings [25, 148, 159]. We take inspiration from Mazloom and Gordon [148, Appendix A], using a shifted and truncated two-sided geometric distribution to sample dummies and achieve approximate DP. In our setting, achieving a reasonable privacy parameter δ requires adding a large number of dummies. Although it improves on the full padding solution, there remains a high communication cost. We elaborate on the details of this protocol and the expected number of dummies and compare it to our final solution in Appendix A.

Now that we have established the drawbacks of some preliminary solutions, we develop a solution which corrects these weaknesses. In our final solution, we reduce the number of dummies and improve the privacy guarantee from approximate to pure differential privacy. Clearly, adding one-sided noise would not allow for pure differential privacy. Intuitively, when only *adding* data and never *removing* data, the output will reveal an upper bound on the true number of data points. Thus, we want to simulate the effects of removing without actually removing data, as that would compromise the accuracy. We accomplish this using

a technique we call *selective multi-party computation*. This involves forwarding each piece of data to a select subset of computation nodes. This simultaneously emulates subtractive noise and reduces the communication costs. We describe this in detail in the following section.

3.5 Protocol Description

This section provides a detailed description of our new protocol designed to compute differentially private statistics over key-value data. Our protocol consists of two phases: data collection and multi-party computation. The main contribution of our work is in the data collection phase. Here, rather than having clients secret share their data among all computation nodes, they only send their shares to a select subset of computation nodes. This atypical approach enables us to improve efficiency and compute differentially private outputs. The multi-party computation phase adapts previously known techniques to account for this change in the data collection phase.

3.5.1 Data Collection Phase

Recall that we have m clients, each denoted by an index $i \in [m]$ and $\ell \geq 3$ computation nodes. We also require an entity for dummy generation, which we assume is semi-honest. In Section 3.5.3, we discuss an alternative approach for dummy generation which does not require a single semi-honest entity. Our protocol aims to return a statistic over the multiset, $\mathcal{S}$, while revealing only differentially private information about any particular client's set of key-value pairs $\mathcal{S}_i$. In the example of collecting telemetry data, this phase corresponds to the step where usage data is sent from users' browsers to the computation servers.

The data flow in this phase is similar to the one-sided dummy solution and is summarized in Figure 3.1.

1. Let $t \in \{2, \dots, \ell - 1\}$ be a (publicly-known) system parameter chosen ahead of time. For each key-value pair belonging to each client $i \in [m]$, the client i chooses a subset of computation nodes of size t , uniformly at random, without replacement. The client generates t shares of their value, one for each computation node they selected, using additive secret sharing. The client also generates t shares of a flag with value 1. This is summarized in Figure 3.2. The client forwards each tuple consisting of the key, a share of the flag, and a share of the value, to the corresponding computation node.

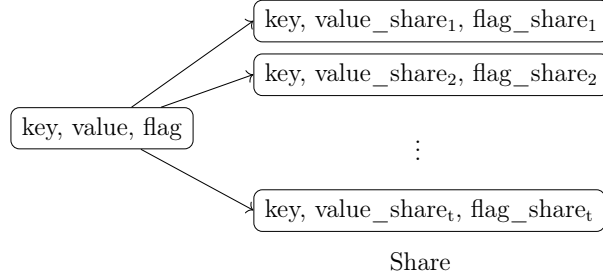


Figure 3.2: Client process for a single key-value pair.

2. Meanwhile, for each key k in the set $\mathcal{K}$, the dummy generator samples x_k from a geometric distribution with parameter r . The two-sided geometric distribution is commonly used as the discrete version of the Laplace mechanism [22, 87]. We use the standard one-sided geometric distribution here to avoid having to shift and truncate the distribution. The dummy generator generates x_k *dummy* key-value pairs with key k , flag 0, and value 0. They repeat the process performed by the clients by choosing a subset of size t of computation nodes uniformly at random and generating t shares of each dummy value and flag. The dummy generator forwards the dummy data to their respective computation nodes.
3. Upon receiving the anonymized data, the computation nodes perform a multi-party computation protocol, based on which statistic is desired, and output the result.

The main difference between this data collection phase and a typical data collection phase in MPC is in step (1) where clients only send their data to a subset of computation nodes. Although conceptually simple, this effectively acts as subtractive noise which will complement the additive noise from the dummy key-value pairs in step (2), thereby ensuring pure differential privacy.

3.5.2 Multi-party Computation Phase

In this section, we describe how the computation nodes process the data once it is received. To compute statistics on the data, they must be able to perform basic operations, such as multiplication and division, and add noise to perturb the final result. A protocol by Wu et al. [199] allows for secure multi-party addition of Laplace noise. We refer readers to the original paper for details of the protocol and proofs of security. Specifically, the protocols allow us to obtain shares of a random variate drawn from a univariate cumulative

distribution function. Additionally, we can use a protocol from Catrina and Saxena to perform division [37, §3.4]. In the following descriptions, we let $[x]$ denote that the value of x is distributed among the parties. We remark that the secret sharing scheme used in the multi-party computation step to generate noise need not be the same additive secret sharing scheme used during data collection; although, a change in schemes may require a *resharing* step to update the shares accordingly. Finally, we note that due to the selective way in which shares are distributed to computation nodes, the types of statistics which may be computed are restricted to those which can be derived from initially taking the sum of all shares. To compute statistics other than those which we describe below, one might consider encoding the data in various ways [45] to achieve compatibility with the selective step of the data collection phase.

Frequency estimation

As per the data collection phase, each computation node receives a subset of shares of key-value pairs. The computation nodes see tuples which contain a key, a share of the flag, and a share of the value. To compute the frequency estimation, the computation nodes compute over the shares of the flag. Recall that real key-value pairs will have a flag with value 1 and dummies will have a flag with value 0 (and this value is not revealed to the computation node since it is secret shared). For each key in $\mathcal{K}$, each computation node executes Protocol 1 with $\Delta = \lambda$, where λ denotes the maximum number of distinct keys held by any client. The value of λ is publicly known. Essentially, Protocol 1 enables the computation nodes to compute the sum of all flags corresponding to some key k and add noise to the result.

Protocol 1 Frequency estimation

Input. A set of triples $\langle k_{ij}, f_{ij}, s_{ij} \rangle$. A target key k . A sensitivity Δ .

1: Compute the sum $f_i = \sum_{k_{ij}=k} f_{ij}$. These are shares of the frequency $[q_k]$.

2: Generate a random variate, $[\xi]$, drawn from $\text{Lap}(\Delta/\epsilon_F)$ by invoking Protocol [199, §7] with $C(t) = \frac{\epsilon_F}{2\Delta} \int_{-\infty}^t \exp(-(\epsilon_F|s|)/\Delta) ds$.

3: Return $[q_k] + [\xi]$.

Correctness. Before the noise addition, the i th computation node has a value f_i . The sum of all of these values is

$$\sum_{i=1}^{\ell} f_i = \sum_{i=1}^{\ell} \sum_{j, k_{ij}=k} f_{ij} = \sum_{k_{ij}=k} f_{ij}.$$

The sum of all shares of the flags must be the sum of the flags themselves by the properties of the additive secret sharing scheme. As real data points have flag equal to 1 and dummies have flag 0, this must be the frequency of key k , q_k . By the properties of Protocol [199, §7], the computation nodes ultimately obtain the frequency of the key k with Laplace noise drawn from $\text{Lap}(\Delta/\epsilon_F)$ where $\Delta = \lambda$.

Mean estimation

To compute a noisy mean estimation, the computation nodes execute Protocol 2 with $\Delta = \frac{\lambda 2R}{\gamma}$, where λ is the maximum number of distinct keys held by any client, γ is the minimum frequency of any key, and $2R$ is the size of the interval bounding the values. To summarize Protocol 2, the computation nodes compute the frequency of the key k , sum the values corresponding to k , divide by the frequency to get the mean, and add noise. Let S_k denote the sum of all values corresponding to key k .

Protocol 2 Mean estimation

Input. A set of triples $\langle k_{ij}, f_{ij}, s_{ij} \rangle$. A target key k . A sensitivity Δ .

1: Compute the sum $f_i = \sum_{k_{ij}=k} f_{ij}$. These are shares of the frequency $[q_k]$.

2: Compute the sum $s_i = \sum_{k_{ij}=k} s_{ij}$. These are shares of the sum of values $[S_k]$.

3: Invoke Protocol [37, §3.4] with input $[S_k], [q_k]$ to obtain $[S_k/q_k]$.

4: Generate a random variate, $[\xi]$, drawn from $\text{Lap}(\Delta/\epsilon_M)$ by invoking Protocol [199, §7] with $C(t) = \frac{\epsilon_M}{2\Delta} \int_{-\infty}^t \exp(-\frac{\epsilon_M|s|}{\Delta}) ds$.

5: Return $[S_k/q_k] + [\xi]$.

Correctness. Steps 1-2 accurately compute the frequency of the key k and the sum of all the values corresponding to key k , by the properties of the additive secret sharing scheme. This is also due to the fact that the dummies have a value of 0, so this does not affect the sum. Step 3 computes the true mean. By the properties of Protocol [199, §7], the computation nodes obtain the mean of the values corresponding to key k with Laplace noise drawn from $\text{Lap}(\Delta/\epsilon_M)$ where $\Delta = \frac{\lambda 2R}{\gamma}$.

3.5.3 Implementation Considerations

Anonymous Channels. In our protocol description, we assume that there exist anonymous channels between the clients and the computation nodes. Although this can be accomplished by using a variety of anonymity systems [59], we discuss one approach here. We may rely on a semi-trusted party to act as an intermediary between the clients and the computation nodes. This is similar to the shuffle model in DP [30], where an intermediary shuffler performs anonymization, shuffling, thresholding, and batching. The presence of the shuffler induces a privacy amplification which allows each client to add less randomness [73]. Our protocol does not require an equally strong trust assumption on the shuffler. Rather, we can have clients encrypt their data with the public keys of the computation node that is responsible for receiving each share of data. Then, the clients forward their data to the shuffler, which removes any user-specific metadata, before forwarding the shares to the respective computation nodes in random order. This ensures that data is anonymized from the perspective of the computation nodes without revealing the data to the shuffler. The only information that the shuffler learns is how many key-value pairs each client holds. Ultimately, our use of this intermediary party requires less trust than the trust required in the shuffle model. Given that the shuffle model is used in practice today by organizations such as Google [73], it is reasonable to assume the existence of anonymous channels. Collusion between the shuffler and the server is not permitted in the shuffle model. Similarly, collusion between the computation nodes and the nodes implementing the anonymous channel is not permitted in our protocol.

In addition to anonymizing data, the shuffler may also be responsible for generating the dummy values. This intermediary party can fulfill two goals, while we retain minimal trust assumptions.

Dummy Generation. Another requirement in the protocol description is a semi-trusted party to generate dummies. A single party that generates dummies poses a threat if that party colludes with the computation nodes. If the total number of dummies generated for key k is known to a computation node, they can derive the expected number of dummies it received for said key. For x_k dummies in total, one computation node receives px_k dummies on average. An estimate of the true frequency of the key is derived by deducting px_k from the total number of key-value pairs it receives.

To circumvent this issue, one alternative is using d parties to generate the dummies. Specifically, for each key k , d parties independently sample x_k from a geometric distribution with parameter r and generate x_k dummies for key k . Collusion between dummy generators and clients does not affect the security of the protocol. Hence, for each key, dummy

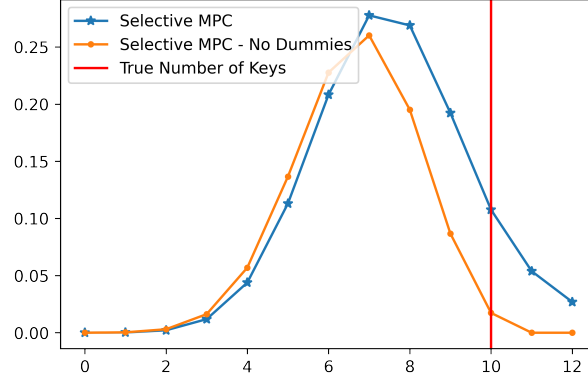


Figure 3.3: We visualize the distribution of a single computation node’s view of 10 keys. We set $p = 2/3$ and $r = 0.4$ and plot the distribution with and without dummies.

generators can be chosen from amongst the clients to eliminate the requirement for extra entities in the protocol. While the total number of dummies that is generated for each key is roughly d times higher than required, this approach distributes trust and prevents a single point of failure. Moreover, our evaluation shows that the expected number of dummies with a single dummy generator is very small (less than 3). Hence, this approach adds a small overhead to the overall protocol. In Section 3.7, we address the case where dummy generators behave maliciously.

Offline Noise Generation The random noise generated in line 2 of Algorithm 1 and line 4 of Algorithm 2 depends only on the parameters of the protocol, not the data provided by the clients. Hence, these steps, which constitute the bulk of the runtime, can be performed offline before the clients provide their data. This reduces the perceived latency from the client’s perspective.

3.6 Privacy

In this section, we analyze the privacy of our protocols. First, we formally define the privacy leakage as a function and show that it is ϵ_L -DP. We find that limiting the view of each computation node by randomly selecting which key-value pairs a node sees is equivalent to drawing noise from a binomial distribution. We visualize this distribution in Figure 3.3 (Selective MPC - No Dummies). In expectation, each node will see $\frac{t}{\ell}$ of the keys ($2/3$ of 10 in the figure), the center of the binomial distribution. Despite being a smooth, two-sided

distribution, the binomial selection alone does not satisfy pure DP. That is because simply distributing the key-value pairs ensures it is impossible for a computation node to observe more than the true count of keys (the probability of 11 or 12 keys is zero in Figure 3.3). For this reason, we add a small number of dummy key-value pairs following a geometric distribution. We can see that this shifts the distribution to the right and avoids clipping the right tail in Figure 3.3. We formalize our unique combination of additive noise from a geometric distribution and subtractive noise from a binomial distribution and prove that it satisfies pure DP. We show in Section 3.6.1 that our frequency estimation satisfies ϵ_F -DP and that our mean estimation satisfies ϵ_M -DP. As a result, the total ϵ spent for the frequency estimation protocol is $\epsilon_F + \epsilon_L$ by naive composition. Similarly, the privacy budget for the mean estimation is $\epsilon_M + \epsilon_L$. The mean and frequency computation can be executed simultaneously, leading to a total cost of $\epsilon_F + \epsilon_M + \epsilon_L$.

Let L be the algorithm that distributes the key-value pairs and dummies among the ℓ computation nodes. Let L_k denote the corresponding algorithm for a single key k . For the sake of the privacy analysis, we treat L as an ideal functionality: we assume it is run by a trusted party and consider only the output of the algorithm in our privacy analysis. An arbitrary output of L is an observation of a single computation node, which we denote by Z . For simplicity, we assume that the nodes do not collude with each other and address the case of collusion separately. Recall that upon receiving data, the computation nodes obtain a set of tuples. The shares reveal nothing about the corresponding values and the computation nodes do not know the origin of the data. Therefore, Z can be described as a histogram of keys observed by the computation node. We show that the output of this algorithm is bounded by differential privacy. The input to L is the list of all key-value pairs held by all users. We say that two inputs, $\mathcal{S}$ and $\mathcal{S}'$, are *neighbouring* if they differ in at most the addition or deletion of λ entries (user-level privacy).

We represent the ground truth of this algorithm as the vector $q(\mathcal{S})$, i.e., the true histogram of the keys. Let $q_k(\mathcal{S})$ represent the count of key k on the input $\mathcal{S}$. We assume that each user has at most one value for each key and thus $\forall k \in [n], |q_k(\mathcal{S}) - q_k(\mathcal{S}')| \leq 1$. The ideal functionality of algorithm L can be described as follows.

1. Compute the noisy histogram Y for a given input $\mathcal{S}$ by computing $Y_k = q_k(\mathcal{S}) + \eta$ where $\eta \in \mathbb{N}$ is sampled i.i.d from a geometric distribution with parameter r .
2. Then Z is obtained by sampling from the histogram Y , following our protocol. This selective part of our procedure is equivalent to sampling $\|Y\|_1$ times from a binomial distribution with parameter $p = t/\ell$.

We denote the probability mass function (PMF) of the binomial distribution¹ with parameters a and p as

$$\mathbb{B}(z; a, p) = \binom{a}{z} p^z (1-p)^{a-z}, \quad (3.4)$$

and the PMF of the geometric distribution as

$$\mathbb{G}(z; r) = (1-r)^z r. \quad (3.5)$$

Then, the PMF of $L_k(\mathcal{S})$ is defined as

$$\Pr[L_k(\mathcal{S}) = Z_k] = \sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_k; q_k(\mathcal{S}) + v, p). \quad (3.6)$$

Finally, the PMF of $L(\mathcal{S})$ is

$$\Pr[L(\mathcal{S}) = Z] = \prod_{k=0}^n \Pr[L_k(\mathcal{S}) = Z_k], \quad (3.7)$$

where this equality follows from the fact that L applies randomness independently for each key. Using this PMF we can prove the following theorem.

Theorem 3.6.1. The algorithm L satisfies ϵ_L -DP with

$$\epsilon_L = \lambda \ln \left(\max \left\{ \frac{1}{1-r}, \frac{1}{1-p} + 1-r \right\} \right). \quad (3.8)$$

Proof. We begin by expanding the statement needed to prove differential privacy. First, we derive

$$\frac{\Pr[L(\mathcal{S}) = Z]}{\Pr[L(\mathcal{S}') = Z]} = \frac{\prod_{k=0}^n \Pr[L_k(\mathcal{S}) = Z_k]}{\prod_{k=0}^n \Pr[L_k(\mathcal{S}') = Z_k]} \leq \left(\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} \right)^\lambda \quad (3.9)$$

where j is one of at most λ keys that differ between $\mathcal{S}$ and $\mathcal{S}'$.

We now consider the two worst cases of neighbouring inputs for a specific key j under our assumption that $|q_j(\mathcal{S}) - q_j(\mathcal{S}')| \leq 1$. These cases are when $q_j(\mathcal{S}) - 1 = q_j(\mathcal{S}')$ and

¹We follow the standard convention that $\mathbb{B}(z; a, p) = 0$ whenever $a < z$.

when $q_j(\mathcal{S}) + 1 = q_j(\mathcal{S}')$. First, we consider the case that $q_j(\mathcal{S}) - 1 = q_j(\mathcal{S}')$. In this case we have that

$$\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} = \frac{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}) + v, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)} \quad (3.10)$$

$$= \frac{\sum_{v=1}^{\infty} \mathbb{G}(v-1; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)}, \quad (3.11)$$

where (3.11) follows from a change of variables $v = v - 1$ and our assumption that $q_j(\mathcal{S}) - 1 = q_j(\mathcal{S}')$. Next we apply the fact that $\mathbb{G}(v-1; r) = \frac{1}{1-r} \mathbb{G}(v; r)$, for $v \geq 1$, to obtain

$$\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} = \frac{\sum_{v=1}^{\infty} \frac{1}{1-r} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)} \quad (3.12)$$

$$\leq \frac{1}{1-r}, \quad (3.13)$$

where the last inequality follows by widening the summation limit to include zero and cancelling like terms. We note that this gives us the first term in the maximum from (3.8).

Next we consider the case where $q_j(\mathcal{S}) + 1 = q_j(\mathcal{S}')$. The first steps proceed similarly except we perform the change of variables $v = v + 1$. That is,

$$\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} = \frac{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}) + v, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)} \quad (3.14)$$

$$= \frac{\sum_{v=-1}^{\infty} \mathbb{G}(v+1; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)} \quad (3.15)$$

$$= \frac{\mathbb{G}(0; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') - 1, p)}{\sum_{v=0}^{\infty} \mathbb{G}(v; r) \mathbb{B}(Z_j; q_j(\mathcal{S}') + v, p)} + (1-r), \quad (3.16)$$

where (3.16) follows from separating off the first term of the sum in the numerator and applying the fact that $\mathbb{G}(v+1; r) = (1-r)\mathbb{G}(v; r)$, for $v \geq 0$, to cancel like terms. Next, we observe that when $Z_j \geq q_j(\mathcal{S}')$, (3.16) simplifies to $(1-r)$ since the binomial coefficient in the numerator is zero. Hence, all that remains is the case where $Z_j < q_j(\mathcal{S}') - 1$. In this case, we get the following inequality by decreasing the upper limit of the sum in the denominator to zero. Specifically,

$$\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} \leq \frac{\mathbb{B}(Z_j; q_j(\mathcal{S}') - 1, p)}{\mathbb{B}(Z_j; q_j(\mathcal{S}'), p)} + (1-r) \quad (3.17)$$

$$= \left(\frac{1}{1-p} \right) \frac{\binom{q_j(\mathcal{S}')-1}{Z_j}}{\binom{q_j(\mathcal{S}')}{Z_j}} + (1-r) \quad (3.18)$$

$$= \left(\frac{1}{1-p} \right) \frac{q_j(\mathcal{S}') - Z_j}{q_j(\mathcal{S}')} + (1-r) \quad (3.19)$$

$$\leq \frac{1}{1-p} + 1-r, \quad (3.20)$$

where (3.18) comes from applying (3.4) and (3.19) applies the definition of the binomial coefficient. Finally, (3.20) follows from the fact that $0 \leq Z_j < q_j(\mathcal{S}') - 1$.

Combining the two cases from (3.13) and (3.20) gives

$$\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} \leq \max \left\{ \frac{1}{1-r}, \frac{1}{1-p} + 1-r \right\}. \quad (3.21)$$

Substituting this result into (3.9) from above, it follows that

$$\frac{\Pr[L(\mathcal{S}) = Z]}{\Pr[L(\mathcal{S}') = Z]} \leq \left(\frac{\Pr[L_j(\mathcal{S}) = Z_j]}{\Pr[L_j(\mathcal{S}') = Z_j]} \right)^\lambda \quad (3.22)$$

$$\leq \left(\max \left\{ \frac{1}{1-r}, \frac{1}{1-p} + 1-r \right\} \right)^\lambda = e^{\epsilon_L}, \quad (3.23)$$

where

$$\epsilon_L = \lambda \ln \left(\max \left\{ \frac{1}{1-r}, \frac{1}{1-p} + 1-r \right\} \right). \quad (3.24)$$

□

The proof relies on the assumption that computation nodes only see a subset of the keys. This simulates adding noise from a binomial distribution. A natural question to ask is how the privacy guarantee degrades under collusion. In the case where computation nodes collude, the privacy degrades.

Claim 3.6.1. *Let c be the collusion threshold which is the maximum number of computation nodes that can collude (Theorem 3.6.1 assumes $c = 1$). Then (3.8) becomes*

$$\epsilon_L = \lambda \ln \left(\max \left\{ \frac{1}{1-r}, \frac{\binom{\ell}{c+1}}{\binom{\ell-c}{c+1}} + 1 - r \right\} \right). \quad (3.25)$$

We note that we consider the minimal value of $t = c + 1$. This is because values of t less than $c + 1$ may allow colluding computation nodes to reconstruct the values.

The result in Equation (3.25) follows from a similar argument to the proof of Theorem 3.6.1, substituting p with the maximum probability that a node observes a specific share in the presence of collusion. The probability of a computation node observing a single share, either directly or through the nodes it colludes with, is $1 - \binom{\ell-c}{c+1} / \binom{\ell}{c+1}$. In Figure 3.4, we show how collusion impacts the privacy budget compared to the baseline given by Theorem 3.6.1. In the baseline, represented by the dotted line, we set $t = 2$, $\ell = 20$, and choose r such that it minimizes ϵ_L . We plot Equation (3.25) as a function of c for $\ell = 20$ and choose r such that it minimizes ϵ_L .

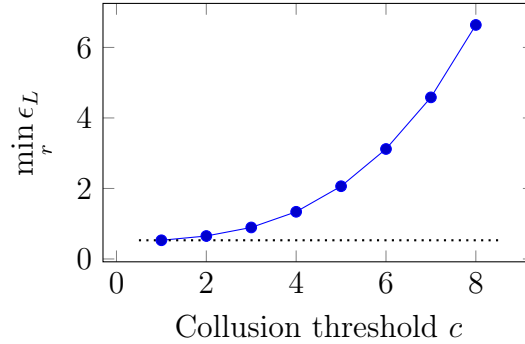


Figure 3.4: ϵ_L for varying levels of collusion.

3.6.1 Privacy of the Output

Frequency Estimation. We denote the output of the frequency estimation protocol by $F(\mathcal{S})$. The output of this algorithm is $q(\mathcal{S}) + \text{Lap}(\Delta/\epsilon_F)^n$, where $\Delta = \lambda$ and $\text{Lap}(\Delta/\epsilon_F)^n$

represents n i.i.d draws from the Laplace distribution with parameter Δ/ϵ_F .

Theorem 3.6.2. The algorithm F satisfies ϵ_F -DP.

Proof. Each client has at most one copy of each key, so we have that $|q_k(\mathcal{S}) - q_k(\mathcal{S}')| \leq 1$. Additionally, we know that each client can have at most λ distinct keys. Hence,

$$\Delta = \max_{\mathcal{S}, \mathcal{S}'} \|q(\mathcal{S}) - q(\mathcal{S}')\|_1 = \lambda. \quad (3.26)$$

The result then follows trivially from the fact that the Laplace mechanism is differentially private. $\square$

Mean Estimation. Recall that the output of the mean estimation protocol is denoted by $M(\mathcal{S})$. Denote the true mean of the dataset $\mathcal{S}$ to be $\mu(\mathcal{S})$. Thus, the output of the algorithm M is $\mu(\mathcal{S}) + \text{Lap}(\Delta/\epsilon_M)^n$ where $\Delta = \frac{\lambda 2R}{\gamma}$ and $\text{Lap}(\Delta/\epsilon_F)^n$ represents n i.i.d draws from the Laplace distribution.

Theorem 3.6.3. The algorithm M satisfies ϵ_M -DP.

Proof. Each client has at most one value for each key and each value is in $[x - R, x + R]$, so it follows that $|\mu_k(\mathcal{S}) - \mu_k(\mathcal{S}')| \leq \frac{2R}{q_k(\mathcal{S})}$. Recall that $\gamma = \min_{k \in [n]} q_k(\mathcal{S})$. Thus, we have that $\forall k \in [n]$, $|\mu_k(\mathcal{S}) - \mu_k(\mathcal{S}')| \leq \frac{2R}{\gamma}$. Additionally, we know that each client has at most λ distinct keys. Therefore,

$$\Delta = \max_{\mathcal{S}, \mathcal{S}'} \|\mu(\mathcal{S}) - \mu(\mathcal{S}')\|_1 \leq \frac{\lambda 2R}{\gamma}. \quad (3.27)$$

Then privacy follows trivially from the fact that the Laplace mechanism is differentially private. $\square$

Outputting Exact Answers. We note that our protocol could be used to output the exact frequency or mean. We would simply follow the selective MPC protocol without adding the final noise (the dummies have no effect on the accuracy). In this case, ϵ_L is the only privacy cost incurred. This is similar to output-constrained DP [98] where the computation nodes do not learn more than a DP amount about the clients data, excluding what they learn from the final result. In general, publishing exact statistics should be avoided as it allows for reconstruction attacks.

3.7 Security

We start by considering the semi-honest model and later extend to the malicious model. Our protocol consists of two main parts: the data collection and the multi-party computation protocols. For simplicity of the implementation, we are working under the assumption that there exist anonymous, secure channels between the clients and the computation nodes. The clients only see their own data and the final output of the differentially private computation. Our goal is that an adversary learns no more than the differentially private leakage from algorithm L and the final output of the differentially private computation.

3.7.1 Proof of security

To analyze the security of the protocol, we rely on a definition of indistinguishable computationally differential privacy (IND-CDP) by Mironov et al. [151]. The following definition is for two-party computation. The security definitions which we present encompass the entire protocol, i.e. both the data collection and computation phases.

Definition 3.7.1 (IND-CDP-2PC [98]). *A two-party protocol Π for computing function f satisfies $(\epsilon_A(\lambda), \epsilon_B(\lambda))$ -indistinguishable computationally differential privacy (IND-CDP-2PC) if $\text{VIEW}_A^\Pi(D_A, \cdot)$ satisfies $\epsilon_B(\lambda)$ -IND-CDP, i.e., for any probabilistic polynomial time (in λ) adversary $\mathcal{A}$, for any neighbouring datasets (D_B, D'_B) ,*

$$\begin{aligned} & \Pr[\mathcal{A}(\text{VIEW}_A^\Pi(D_A, D_B)) = 1] \\ & \leq \exp(\epsilon_B) \cdot \Pr[\mathcal{A}(\text{VIEW}_A^\Pi(D_A, D'_B)) = 1] + \text{negl}(\lambda). \end{aligned}$$

Likewise for B 's view of any neighbours (D_A, D'_A) and ϵ_A .

For our purposes, we extend the definition from two-party to multi-party computation and we let $\epsilon = \epsilon_A = \epsilon_B$ for convenience. Intuitively, the following definition ensures that an adversary $\mathcal{A}$ cannot distinguish neighbouring databases D, D' from their view, and the same applies to every other party.

Definition 3.7.2 (IND-CDP-MPC). *A multi-party protocol Π for computing function f satisfies $\epsilon(\lambda)$ -indistinguishable computationally differential privacy (IND-CDP-MPC) if for every probabilistic polynomial time (in λ) adversary $\mathcal{A}$ with input D_A , and for neighbouring datasets D, D' belonging to the honest parties (i.e. $D, D' = \cup_{i \in \mathcal{A}} D_i$),*

$$\begin{aligned} & \Pr[\mathcal{A}(\text{VIEW}_A^\Pi(D_A, D)) = 1] \\ & \leq \exp(\epsilon) \cdot \Pr[\mathcal{A}(\text{VIEW}_A^\Pi(D_A, D')) = 1] + \text{negl}(\lambda). \end{aligned}$$

Likewise for every other party's view of neighbours (D, D') and ϵ .

Let Π denote the complete protocol, from the clients sending data to the computation nodes and the computation nodes performing the multi-party computation protocol to compute a function f over the data. Our security property will apply to multi-party computation protocols which satisfy *statistical privacy*. We say that the MPC protocol computing f ensures *statistical privacy* if the views of the computation nodes can be simulated by an oracle such that the true and simulated distributions are statistically indistinguishable. Both our protocol computing the frequency estimation and the protocol computing the mean estimation satisfy statistical privacy. Wu et al. [199] prove that their noise generation protocol satisfies statistical privacy and Catrina and Saxena [37] prove that their division protocol satisfies statistical privacy. The remaining steps in our protocols are secure in an information-theoretic sense. Thus, the following security proof can be applied to both our frequency and mean estimation protocols.

In the definition that we aim to satisfy, the adversary controls some set of parties, denoted by A . The adversary we consider may control at most b_N computation nodes, where b_N will be defined depending on the scenario. Then, their view consists of the combined view of the b_N computation nodes which they control. Note also that an adversary may control up to $b_C \leq m - 1$ clients (all but one). Since a client does not perform any computation and simply submits data, their view is trivial to simulate. Further, we assume for the time being that the protocol is using a trusted third party to generate dummies, which cannot be controlled by the adversary. In order to determine b_N , the bound on the number of computation nodes that the adversary may control, we must consider how the MPC protocol functions. In our case, we use a (t, t) -threshold scheme, so we can let $b_N = t - 1$. Then, the neighbouring datasets, D and D' , correspond to the data sent by the honest clients which are outside of the control of the adversary. Finally, we state our security theorem and defer the proof to the full paper [102, Appendix C].

Theorem 3.7.1. If f satisfies ϵ_f -differential privacy and the corresponding MPC protocol satisfies statistical privacy, then our protocol Π ensures IND-CDP-MPC in the presence of a semi-honest adversary who controls at most b_N computation nodes and at most b_C clients.

3.7.2 Extending to the malicious model.

Our security proof assumed that we work in the semi-honest model, in which all parties follow the protocol. We now consider how our protocol may function if this assumption is relaxed. First, consider the possibility of malicious computation nodes. In the data

collection phase, we require that the computation nodes act semi-honestly so that they are unable to arbitrarily add their own input. We can relax this assumption in the multi-party computation phase. Methods for extending MPC protocols to be secure in the presence of malicious parties have been discussed in the literature [51, 117]. We can apply the same strategies to our own work, using protocols that are maliciously secure under the assumption that there exists an honest majority [29, 51, 67]. This prevents nodes from adding arbitrary noise during the computation phase which may affect the results. In this scenario, we can adjust our bound b_N on the number of computation nodes that the adversary may control to be the minimum of $t - 1$ and $\lfloor (\ell - 1)/2 \rfloor$. This ensures that the adversary cannot control a majority of participants in the multi-party computation. As with any other multi-party computation protocol which assumes an honest majority, if the assumption of an honest majority is not satisfied then there are no guarantees on the privacy or correctness of the output. In this paper, we have discussed how our protocol can account for semi-honest computation nodes and a minority of misbehaving computation nodes; however, we are not limited to these two settings. Since the multi-party computation phase is a simple adaption of previously known techniques in MPC, it is straightforward to substitute an MPC protocol with other guarantees. For example, in a scenario where one is concerned about the reconstruction of invalid or incorrect results, we can use an MPC protocol which ensures robustness. Similarly, we could use a protocol which is secure against Byzantine failures if we are concerned about computation nodes failing. The protocol which is ultimately chosen depends on the requirements of the system in practice, but is not limited in any way by our construction.

Next, we consider the possibility of malicious clients. We can prevent malicious clients from submitting invalid data by having the computation nodes perform validation checks on the shares they receive. Note that the validity check is conducted in MPC by the t computation nodes that receive each piece of data, not all ℓ computation nodes. For example, when t computation nodes receive the shares of some flag x , they can compute $x \cdot (1 - x)$ to check that the result is 0 (which we evaluate in Section 3.8.3). This verifies that the flag is in the set $\{0, 1\}$. The computation nodes learn no additional information by computing validation checks with the other nodes which receive shares of the same data. Similar checks can be computed on the data received from the clients to ensure that it exists in some valid range. Thus, we are able to ensure that clients are submitting valid data within some pre-defined, acceptable range. Validity checks other than those described here can also be implemented and our protocol does not impose any restriction on input validation. These verification checks can be implemented in the setting where there is a dishonest majority among the subset of computation nodes which receive shares by using MPC protocols like SPDZ [78].

Finally, suppose that the third party generating dummies is malicious. On one hand, a malicious dummy generator may submit additional data points to attempt to influence the final result. This poses no more of a threat than the case where malicious clients submit false data points. This is a potential vulnerability to any system which collects data from clients and is not prevented by security against malicious adversaries. In such a scenario where the dummy generator is malicious, it is also possible that they submit no dummies or submit an incorrect number of dummies. This would compromise the differential privacy guarantees on the leakage of our protocol, but would not compromise the correctness of the output. It is difficult to ensure that dummy generation by a single party occurs correctly. However, recall that we have discussed alternatives to dummy generation which provide more distribution of trust in Section 3.5.3. Suppose we are using the method described in Section 3.5.3 where several clients are responsible for generating dummies. Then, we must ensure that the adversary cannot control all of the clients generating dummies. If we assume that the adversary can control at most b_C clients, then we can let $b_C + 1$ clients generate dummies, which ensures that we still achieve differential privacy. This does not affect the security proof because the adversary is unable to control at least one party generating dummies. Ultimately, distribution of trust allows us to consider the presence of malicious parties while achieving secure dummy generation.

3.8 Evaluation

This section provides a theoretical evaluation of our protocol in terms of accuracy, communication, and computation cost. We then compare this with other methods for computing statistics on key-value data. Finally, we present our experimental results to demonstrate the practicality of our protocol.

3.8.1 Our Protocol

For the purpose of comparison, we analyze our protocol in the case where $t = 2$. This only affects the communication cost during the data collection phase and is a reasonable choice in practice as it minimizes ϵ_L .

Communication. The communication conducted in our protocol can be decomposed into the communication from the clients (and dummy generator) to the computation nodes, i.e. data collection, and the communication between computation nodes. For comparison, we

focus mainly on client-to-server communication. The communication between computation nodes is specific to solutions using MPC and depends largely on parameters of the MPC protocol, such as the precision of the noise generated. These are chosen by a system administrator depending on the context of the situation. We provide some experimental results for communication between computation nodes in Section 3.8.3. In this section, we elaborate on the communication between clients and servers. First, $2|\mathcal{S}|$ elements are sent from the clients to the computation nodes because each client generates $t = 2$ shares for each of the key-value pairs it owns. The number of dummies added for each key follows a geometric distribution with an expected value of $\frac{1}{r}$, which means $\frac{n}{r}$ dummies are generated on average. For each dummy, $t = 2$ shares are generated. Therefore, in expectation, $\theta(|\mathcal{S}| + \frac{n}{r})$ shares are sent from the clients (and dummy generator) to the computation nodes.

Computation. We define the computation cost of our protocol to be the number of multiplications in the MPC protocol. For each key, the noise generation (and division in the case of mean estimation) is run to derive the result. The complexity of these operations depends on the chosen protocol and parameters related to their implementation, such as precision. Hence, we consider the complexity to be constant with respect to each key. Therefore, the overall number of multiplications in the MPC protocol across all keys is $O(n)$.

Accuracy. Finally, we consider the accuracy of the output of our protocol. Our protocol implements an algorithm with output identical to that of the central model, so it suffices to consider the accuracy of the Laplace mechanism, which is well known. We begin by evaluating the accuracy of our frequency estimation protocol denoted by $F(\mathcal{S})$. Using a tail bound on the Laplace distribution along with the union bound as shown in Theorem 3.8 of Dwork and Roth [70], we obtain the following accuracy guarantee $\forall \beta \in (0, 1]$.

$$\Pr[||q(\mathcal{S}) - F(\mathcal{S})||_\infty \geq \ln(n/\beta) \cdot (\lambda/\epsilon_F)] \leq \beta. \quad (3.28)$$

Similarly, for the mean estimation protocol, with output denoted $M(\mathcal{S})$, we obtain that $\forall \beta \in (0, 1]$,

$$\Pr[||\mu(\mathcal{S}) - M(\mathcal{S})||_\infty \geq \ln(n/\beta) \cdot (2R\lambda/\epsilon_M\gamma)] \leq \beta. \quad (3.29)$$

To allow for comparison of different methods, we simplify the above expression and obtain an approximation for the error of our protocols using the standard deviation. For a single key, we compute that the error is approximately $O(\Delta/\epsilon)$ where Δ is the sensitivity

	Client-to-Server Communication	MPC Complexity	Error
Our Protocol	$\theta(\mathcal{S} + \frac{n}{r})$	$O(n)$	$O(\Delta/(\epsilon - \epsilon_L))$
Central DP	$O(\mathcal{S})$	-	$O(\Delta/\epsilon)$
Naive MPC	$O(\mathcal{S} \ell)$	$O(\mathcal{S} n)$	$O(\Delta/\epsilon)$
PrivKVM	$O(m \log n)^*$	-	$O(\sqrt{m}\Delta/\epsilon)^*$
PCKV	$O(m \log n)$	-	$O(\sqrt{m}\Delta/\epsilon)$

Table 3.2: Comparison of client-server communication, computation (number of multiplications), and error (approximate bound). *Includes a multiplicative factor equal to the number of iterations which we exclude.

of the specific protocol. For simplicity, we will only consider the error for a single key’s frequency or mean and note the case of all keys can easily be obtained using a union bound. For our analysis, we assume all clients hold this key. Taking into account the privacy budget from the leakage function, we conclude that our solution achieves an approximate error bound of $O(\Delta/(\epsilon - \epsilon_L))$.

3.8.2 Comparison with Other Work

We now compare the communication, computation, and accuracy of our protocol with a naive MPC solution that conceals and secret shares keys, existing local DP solutions (PrivKVM [204], PCKV [94]), and the central model. The results of our comparison are summarized in Table 3.2. Although there has been no prior work on key-value data in the shuffle model, we discuss potential solutions in this setting in Section 3.10.

Naive MPC A naive application of MPC would have each client secret share keys and values and forward the shares to each computation node. This requires no dummies; however, the computation nodes cannot determine which shares correspond to which key. Each time they want to perform a computation over a key-value pair, they need to collectively index into a table containing all the keys. This results in an additional $|\mathcal{S}|n$ multiplications in the MPC protocol. Since clients create a share for each computation node, the client-to-server communication cost is $O(|\mathcal{S}|\ell)$. This approach simulates the central model, so it has an approximate error of $O(\Delta/\epsilon)$.

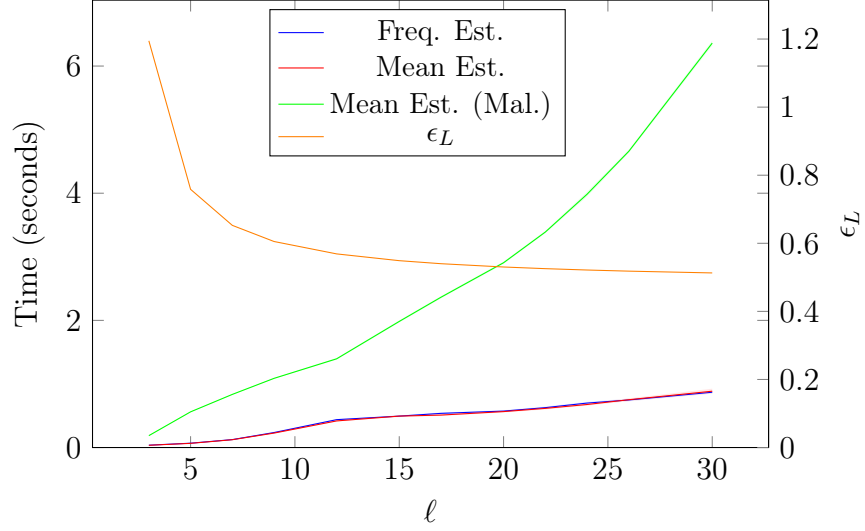


Figure 3.5: Run time for a single key. Measurements are means with 95% confidence interval in shaded region. ϵ_L is shown for the number of computation nodes.

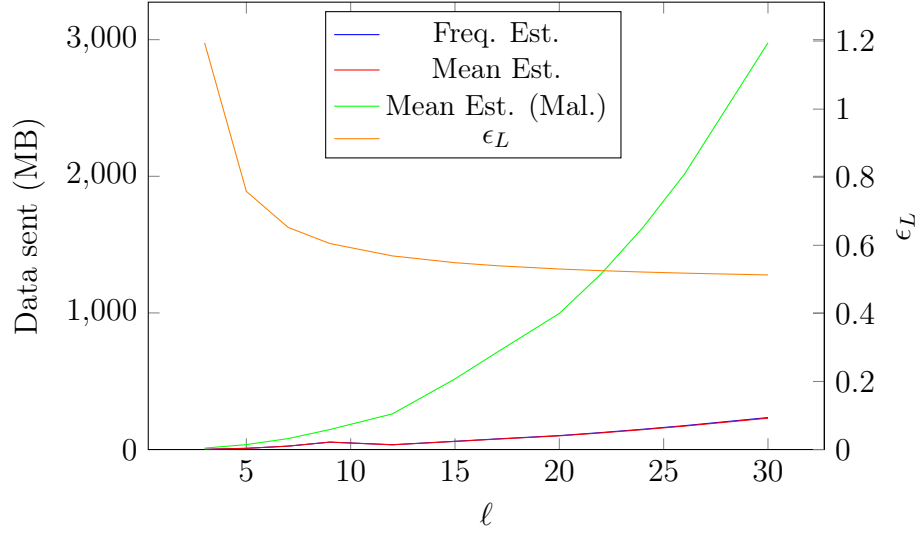


Figure 3.6: Communication for a single key. Measurements are means with 95% confidence interval in shaded region. ϵ_L is shown for the number of computation nodes.

PrivKVM PrivKVM uses an LDP protocol in which each client sends a single perturbed key-value pair to the data curator. Each client samples this key-value pair randomly from their set of key-value pairs. This constitutes all the communication in the protocol. Thus, using our notation for the size of the key domain and the number of users, the total communication complexity of this protocol is $O(m \log n)$. The curator calculates the desired statistics (frequency and mean) by iterating over the data they received and calibrating the result. This computation cost is negligible in comparison to an MPC protocol. Finally, PrivKVM utilizes the Harmony protocol [165] and achieves an approximate error bound of $O(\sqrt{m}\Delta/\epsilon)$. We remark that the accuracy depends on the number of iterations c carried out in the PrivKVM protocol. We follow the approach taken by the authors and treat c as a constant in our analysis. Additionally, there is a sampling error due to the fact each client only sends one of their keys pairs which we omit from this analysis.

PCKV This protocol improves upon PrivKVM. To estimate the mean, each client sends either a key-value pair or a vector equal to the size of the key domain. For our comparison, we will consider the former which has a lower overall communication complexity of $O(m \log n)$ across all users. The protocol does not require interaction, so there is no multiplicative factor for the number of rounds. In terms of accuracy, all of the improvements influence the error by a constant amount. While this showed significant improvements in experimental evaluations, it has no asymptotic advantage over PrivKVM.

3.8.3 Experimental Results

To evaluate the practicality of our protocol, we compute a series of benchmarks. Recall that our protocol consists of two phases: the data collection and multi-party computation phase. The bottleneck of the data collection phase is the communication of data from all clients. Collecting shares is as simple as sending a single message (to several nodes). This overhead is shared by all other related work and a detailed comparison of the size of the communication is given in Table 3.2. Moreover, the bulk of the overhead in the data collection phase consists of the time required for coordination among the clients, i.e., the time for all clients to send their data, which is not reflected in experiments. The computational overhead of our system in the data collection phase is the dummy generation which incurs a negligible overhead equivalent to sharing a few additional key-value pairs (we evaluate exactly how many in Appendix A.4). The most expensive part of our protocol is the MPC phase which requires tightly synchronized nodes. Thus, we implement this phase using the noise generation protocol from Wu et al. [199] and provide benchmarks for

the run time and communication. We investigate the effects of the number of computation nodes, network latency, and the number of keys. Our implementation for the evaluation is publicly available.²

Setup We use the MP-SPDZ [117] framework to implement Protocol [199, §7] from Wu et al. for generating a Laplace random variate. We simulate the frequency or mean estimation protocol by sampling from the Laplace distribution and adding it to an arbitrary result (either a constant or a division of constants). We use the Shamir protocol, implemented in the MP-SPDZ framework.³ Using this implementation, we observed that the underlying technique for generating Beaver triples switches when the number of computation nodes exceeds a threshold. The default threshold was set too high for our use case, resulting in sub-optimal run time results. We set the threshold to ten computation nodes to optimize run time results; however, there remains a noticeable change in the communication results.

Number of Comp. Nodes First, we measure how our protocol scales with the number of computation nodes. Since our protocol can be parallelized over the number of keys, we only consider a single key for this benchmark. To create multiple parties, we use up to 30 Amazon EC2 T2 instances. Each machine has four cores, 16 GB of memory, and is located in the same Ohio data center.

In Figure 3.5, we plot the average run time over 100 runs with 95% confidence intervals as the number of nodes increases for frequency and mean estimation in the semi-honest model and mean estimation in the malicious model. For context, we also plot the corresponding ϵ_L for each ℓ . We see roughly quadratic growth in the run time, which is consistent with the asymptotic complexity of MPC. In the mean estimation protocol, for 30 nodes (where $\epsilon_L = 0.512$) we observe 0.88 and 6.36 seconds for the semi-honest and maliciously secure protocol, respectively. Moreover, for 5 nodes (where $\epsilon_L = 0.758$) the protocol runs in 0.07 and 0.83 seconds in the semi-honest and malicious model, respectively. Figure 3.6 shows the corresponding total communication. The dip around ten nodes comes from the change in protocol mentioned in the previous section. In the mean estimation protocol, we observe 229.45 MB and 2976 MB for 30 nodes in the semi-honest and malicious model and 8.77 MB and 36.11 MB for five nodes in the semi-honest and malicious model. We conclude that our protocol scales acceptably to realistic numbers of computation nodes.

²<https://git.uwaterloo.ca/r5akhava/selective-mpc>

³<https://github.com/data61/MP-SPDZ>

Different Data Centers Having all computation nodes in the same data center is not always possible in practice. Thus, in this experiment, we consider the effects of network latency from having computation nodes in different locations. To do this, we consider three different scenarios: local, remote, and distant. In the local scenario, all nodes are in the Ohio data center. In the remote setting, half of the nodes are in the Ohio data center and half in Northern California. Finally, the distant scenario has half of the nodes in the Ohio data center and the other half in Frankfurt, Germany. We consider a single key and conduct this experiment for 3, 6, 10, and 20 computation nodes. Moreover, we experiment with a semi-honest and malicious secure MPC protocol. The results are given for 100 runs in Table 3.3. We observe reasonable run times even for a high number of computation nodes. Despite increased latency in wide area networks, the total run time remains under six seconds.

Varying Key Domain Sizes Furthermore, we investigate the effects of multiple keys. Recall that each key is treated separately in our protocol, so we parallelize the computation over the keys. To do this, we use the built-in threading functionality in MP-SPDZ. We use five computation nodes (which achieves $\epsilon_L = 0.758$), each running an Amazon EC2 C5 instance with 64 cores and 128 GB of memory. We consider key domain sizes of 10, 100, 1000, and 10,000. As with the previous experiment, we use local, remote, and distant network setups. Results are given for 40 runs in Table 3.4. We observe reasonable run times that increase approximately linearly after the number of keys exceeds the number of cores.

Input Validation We examine the cost of validating the client’s input. We implement the $x \cdot (1 - x) \stackrel{?}{=} 0$ check described in Section 3.7.2, but this can easily be extended to checking arbitrary constraints. Table 3.5 shows the time to perform input validation using 3 computation nodes. The input validation can be performed in parallel so we conduct the experiment for 10, 100, 1000, and 10,000 input data points using the Amazon EC2 C5 instances described in Section 3.8.3. Similar to the other experiments, we use local, remote, and distant network setups. The runtimes for input validation remain reasonable, particularly compared to the runtime of the computation phase, which suggests that input validation does not impose an impractical burden on the protocol.

3.9 Discussion

Privacy Limitations An interesting property of our protocol is that for a fixed choice of ℓ , there is a lower bound on the leakage privacy ϵ_L . This differs from other common DP

# Nodes	ϵ_L	Local	Remote	Distant
3	1.19	0.037 (0.002)	2.323 (0.020)	4.50 (0.52)
		0.19 (0.04)	2.91 (0.06)	5.66 (0.12)
6	0.69	0.077 (0.004)	2.65 (0.15)	4.80 (0.12)
		0.59 (0.03)	3.55 (0.11)	6.38 (0.14)
10	0.59	0.237 (0.047)	3.01 (0.12)	5.85 (0.27)
		1.12 (0.05)	4.34 (0.18)	7.53 (0.24)
20	0.53	0.573 (0.064)	3.18 (0.14)	5.73 (0.22)
		2.91 (0.09)	6.68 (0.27)	10.17 (0.60)

Table 3.3: Run time (in seconds) for one key. Measurements are means with standard deviations in parentheses. For each configuration the first row and second row use a semi-honest and maliciously secure MPC, respectively.

# Keys	Local	Remote	Distant
10	0.075 (0.003)	2.58 (0.09)	5.015 (0.08)
	0.15 (0.00)	3.10 (0.07)	6.08 (0.43)
100	0.58 (0.03)	5.0 (0.3)	9.4 (0.4)
	1.34 (0.11)	5.64 (0.37)	10.64 (0.27)
1000	2.38 (0.03)	32.52 (0.87)	63.6 (1.1)
	3.59 (0.06)	34.28 (0.94)	65.86 (0.38)
10000	20.00 (0.07)	307.98 (8.09)	602.3 (5.6)
	26.64 (0.09)	323.65 (0.23)	627.30 (0.39)

Table 3.4: Run time (in seconds) for many keys with 5 computation nodes, which corresponds to $\epsilon_L = 0.758$. Measurements are means with standard deviations in parentheses. For each configuration the first and second row use a semi-honest and maliciously secure MPC, respectively.

# Inputs	Local	Remote	Distant
10	0.08 (0.00)	1.01 (0.07)	2.02 (0.13)
100	0.80 (0.03)	1.31 (0.06)	2.38 (0.12)
1000	0.73 (0.02)	2.02 (0.09)	3.75 (0.16)
10000	0.86 (0.05)	9.47 (0.15)	18.34 (0.14)

Table 3.5: Run time (in seconds) to validate multiple inputs with 3 computation nodes using maliciously secure MPC. Measurements are means with standard deviations in parentheses.

mechanisms where the privacy budget can be made arbitrarily small, albeit with detrimental effects on the utility. We assume ℓ has a constant value because it is typically a fixed aspect of the setup. In contrast, r is easily modifiable so we can choose $r \in (0, 1)$ to minimize ϵ_L , where r is the parameter for the geometric distribution. Figure 3.7a plots the leakage privacy as a function of r for three fixed values of ℓ based on the result of Theorem 3.6.1. The lower bound on ϵ_L , which arises from the max function in the formula for ϵ_L , can be seen in the graphs. Specifically, for a given ℓ , the smallest achievable ϵ_L can be derived as follows.

$$\min_{r,t} \epsilon_L = \min_{r,t} \left\{ \Delta \ln \left(\max \left\{ \frac{1}{1-r}, \frac{1}{1-\frac{t}{\ell}} + 1 - r \right\} \right) \right\} \quad (3.30)$$

$$= \Delta \ln \left(\frac{2(1 - 2/\ell)}{\sqrt{1 + 4(1 - 2/\ell)^2} - 1} \right) \quad (3.31)$$

which is achieved at $t = 2$ and

$$r = 1 - \frac{\sqrt{1 + 4(1 - 2/\ell)^2} - 1}{2(1 - 2/\ell)} \quad (3.32)$$

Figure 3.7b graphs the smallest achievable privacy of the leakage function based on Equation (3.31). The minimum of Equation (3.31) is obtained at $\ell = \infty$, albeit impossible to achieve in practice, which gives a lower bound on ϵ_L . In particular, if $\Delta = 1$ for simplicity, then

$$\epsilon_L > \min_{\ell} \ln \left(\frac{2(1 - 2/\ell)}{\sqrt{1 + 4(1 - \frac{2}{\ell})^2} - 1} \right) = \ln \left(\frac{2}{\sqrt{5} - 1} \right) = 0.4812. \quad (3.33)$$

Such a lower bound may be unacceptable in certain scenarios. Tightening the analysis of Theorem 3.6.1 would improve this bound.

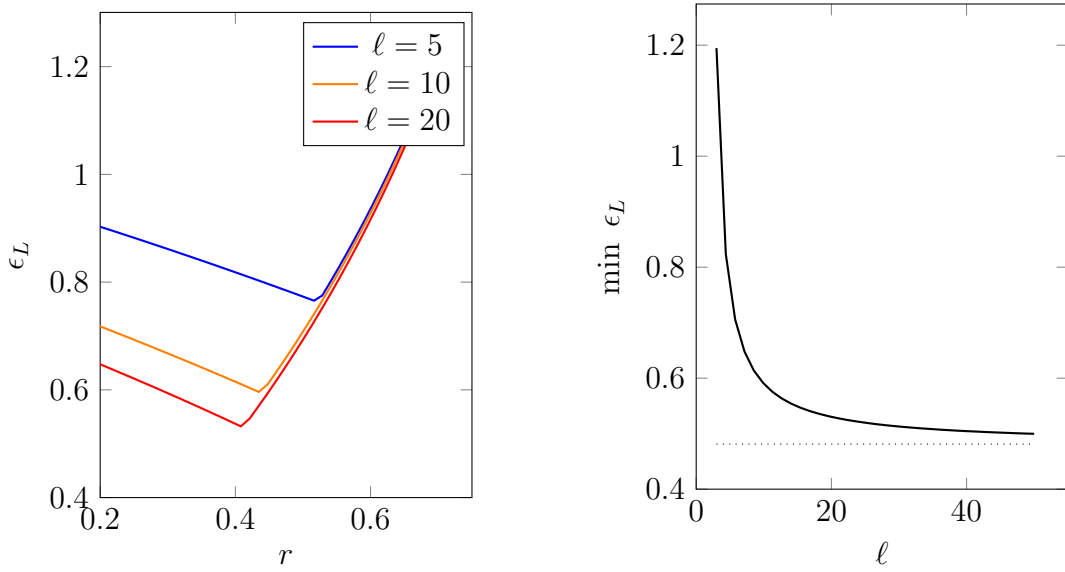


Figure 3.7: (a) Leakage privacy as a function of r for $t = 2$. (b) Lower bound on leakage privacy.

3.10 Related Work

Differentially Private Statistics over Key-Value Data Our work is most closely related to prior literature computing key-value analytics under the LDP model. The first work in this space was PrivKV [204]. Ye et al. modify a randomized response based protocol [165] to better maintain the relationships between the keys and values. They apply their baseline technique in an iterative manner, called PrivKVM, where each iteration gets closer to the true mean. To reduce communication costs, PrivKVM requires each client only send a single randomly sampled key-value pair. This sampling comes at the cost of

additional sampling error in the final result. Follow-up work from Gu et al. [94] introduced PCKV, which improves upon PrivKVM. They apply an advanced sampling procedure to enhance utility over the sampling done by PrivKVM. They also require only a single iteration and provide a tighter analysis of the privacy budget consumption. Sun et al. [188] investigate alternate perturbation techniques under the PrivKV framework and conditional analysis of key-value data.

While this line of work has made substantial advancements to the computation of key-value statistics in the local trust model, it suffers an inherent limitation. In any LDP solution, the randomness is introduced to each value before aggregation and thus the error compounds in the final result. As we show in our evaluation, this increases the error by a factor of $O(\sqrt{m})$. Bittau et al. [30] investigate reducing the effects of compounding noise by introducing the shuffle model of DP. The shuffle model lies in between the local and central models, with a weaker trust assumption than the central model but improved utility guarantees over the local model. In this model, a semi-trusted shuffler lies between the participants and the untrusted curator. The shuffler introduces anonymity by performing a permutation on the noisy data it receives from the participants before handing it to the curator. Since the initial work from Bittau et al. [30], there have been many interesting works in this space [23, 43, 73]. Although these works significantly reduce the dependency on the number of clients (for example Balle et al.’s work has a dependency of $O(\sqrt[6]{m})$ [23]), they still involve adding noise locally, which in addition to making the error dependent on the number of clients, disrupts the correlation between keys and values. It has been shown that the utility of this approach is strictly between that of the local and central models [23, 43]. Thus in our work, we take the approach of simulating the central trust model using MPC to avoid any dependence on the number of clients by adding randomness in a central manner.

Combining Cryptography and Differential Privacy The area of research dedicated to the intersection of cryptography and differential privacy has been termed DP-Cryptography [197]. One direction of this research uses MPC when computing DP statistics to eliminate a trusted central data curator [67, 71, 159]. Dwork et al. [67] provided protocols to generate Gaussian and exponentially distributed noise in a distributed setting. The combination of secure computation and DP was further explored in PrivaDA [71], a generic design that allowed for noise generation in a fully distributed setting. We note that these constructions cannot immediately be applied to key-value data without incurring high computation costs as a result of secret sharing the keys. Pettai and Laud [171] implemented the DP sample-and-aggregate method in an MPC framework, showing that the combination can still provide reasonable performance. Later work used homomorphic encryption for the

secure computation protocols [13, 129, 175, 180]. Goryczka et al. [92] implemented and compared early models that combined MPC and DP.

Another area of DP-Cryptography uses DP to improve the performance of cryptographic primitives. Allowing for differentially private leakage can improve the computation or communication costs. Chen et al. [42] employ this technique in the context of searchable symmetric encryption. Toledo et al. [191] relax the information-theoretic requirements of private information retrieval to provide more efficient constructions which satisfy differentially private leakage. Similarly, Shrinkwrap [25] uses DP query processing to improve the performance of evaluating SQL queries. Mazloom and Gordon [148] extend the concept of differentially private leakage to secure multi-party computations of histograms, PageRank, and matrix factorization. DP leakage has also been used in the context of anonymous communication to reduce communication costs [126, 195, 196]. Our work is unique in that we simultaneously take both approaches to combining DP and cryptography: we allow differentially private leakage to improve performance and use secure computation to eliminate trust assumptions.

3.11 Conclusion

Although local differentially private solutions are becoming increasingly popular, they are also well known to provide poor utility. Unfortunately, a common way to circumvent this issue is to let ϵ be unreasonably large, offering almost meaningless protection to users' privacy. By minimizing the costs that organizations incur when choosing privacy-aware solutions, we can encourage the development and use of systems that respect users' privacy.

In line with this ideal, we provide an improved solution for calculating the frequency and mean of key-value data in the local trust model. Our work presents a novel approach to MPC, which circumvents traditional efficiency issues. Simultaneously, our approach enables us to achieve pure differential privacy guarantees on leakage without removing any data. By leveraging cryptographic primitives, we maintain the correlation between keys and values, resulting in a high accuracy algorithm. We believe that offering high efficiency, accuracy, privacy, and conceptual simplicity incentivizes the adoption of our approach in practice.

Chapter 4

PE-means: Improved Differentially Private k -means Clustering through Private Evolution

This chapter also appears as preprint available on arXiv [105]. We consider a system where a trusted party has a dataset of sensitive user records in plaintext and wishes to cluster the dataset into k sets, publishing the cluster centres. To protect the participants in the dataset, we assume the trusted party will randomize the clustering process to ensure differential privacy. The challenge is that adding privacy to the clustering algorithm reduces the utility of the final result. If the resulting clusters do not sufficiently represent the trends in the dataset, the private version will not be deployed. In this chapter, we propose a new approach that improves the clustering utility over related work while maintaining the same theoretical level of privacy.

4.1 Introduction

The k -means clustering algorithm is a foundational tool in data science with many influential applications such as recommendation systems and healthcare data analysis [86, 150]. The objective is to group the data into k groups with similar features, allowing practitioners to interpret complex datasets. The challenge is that applying standard k -means algorithms to sensitive data risks the privacy of participants in the dataset. In the worst case, the k -means algorithm can publish the exact data record of an individual who is an outlier in

the set. To address this, there has been a large body of work that considers the problem of differentially private k -means [21, 32, 39, 55, 79, 84, 150, 185, 186]. Differential privacy (DP) is a popular privacy definition that ensures a change in a single user’s input does not have a significant effect on the output of the mechanism [69]. By adding calibrated randomization to the clustering process, existing approaches retain the benefits of k -means clustering while providing a formal privacy guarantee.

The challenge with DP is balancing the inherent trade-off between privacy and utility. Existing state-of-the-art approaches for DP k -means rely on summing the private data directly at some point in the process [21, 39, 186]. The sensitivity of (or largest impact any one user can have on) a sum query is determined by the addition (or removal) of a point on the boundary of the domain. This means that to satisfy DP, noise must be added proportionally to the entire domain, significantly degrading the quality of the clustering. In recent work, an approach called FastLloyd managed to reduce this sensitivity from the whole domain to a smaller radius around each cluster through relative cluster updates [55]. However, the radius is still proportional to the data dimension, and FastLloyd introduces an additional error term due to the relative updates.

Private evolution (PE) is an increasingly popular technique for generating various modalities of synthetic data [134, 135, 193, 201]. PE evolves a population of synthetic data using only inference API access to foundation models that generate and perturb the data in an evolutionary algorithm. A key component in PE’s success is that it only uses the private data in a nearest neighbour voting scheme to choose the best samples generated so far. Specifically, each private data point only contributes a single vote to the algorithm in each iteration. Thus, adding or removing any private data point affects the output by a constant sensitivity of 1, regardless of the data domain. This gives PE a significant advantage over gradient-based techniques, which must add noise proportionally to the gradient domain. What makes PE truly impressive is that, despite getting much less signal from the private data, it still effectively traverses large and complex data domains by leveraging the robust optimization characteristics of evolution.

In this work, we introduce PE-means, an extension of the PE framework to the problem of k -means clustering. In this application, the population of PE contains randomly generated centroids that are iteratively evolved towards the centroids of the private data. Our design replaces PE’s API calls to foundation models with a lightweight clustering initialization and a Lévy flight-based mutation operator. In addition to adapting PE to clustering, we also make several improvements to the algorithm that are of independent interest. Specifically, we address a critical issue with PE’s selection technique where votes can be split between the best candidates, causing no representative from certain areas to be selected. We also reduce the impact of DP noise through better post-processing of the vote histogram

and a mechanism to adaptively adjust the signal-to-noise ratio at no cost to privacy. Our experimental evaluation shows PE-means outperforms state-of-the-art baselines over numerous real and synthetic datasets. Furthermore, we propose HDPE-means, a variant of PE-means that uses a similar dimensionality reduction to Balcan et al.’s [21], allowing PE-means to scale to higher dimensions. Overall, we observe up to 91% improvement in clustering loss over the state-of-the-art, with an average improvement of 26% across all the datasets evaluated.

4.2 Background

4.2.1 k -Means Clustering

Given a dataset D of size N and dimension d , the k -means problem aims to partition the dataset into k groups (clusters) $C = \{C_1, \dots, C_k\}$, with each $C_i \subset D$, such that the following objective is minimized,

$$\arg \min_C \sum_{i=1}^k \sum_{x \in C_i} \|x - \mu_i\|^2 \quad (4.1)$$

where μ_i is the centroid of cluster i ($\mu_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$). In this work, we assume the size and dimension of the dataset are public information, but we require any other information published from the dataset to satisfy differential privacy.

4.2.2 Differential Privacy

Differential privacy (DP) [69] adds randomness to the computation of aggregate statistics to protect the privacy of individuals. DP guarantees that an algorithm’s output is approximately the same, regardless of the participation of any one user. More formally, differential privacy can be defined as follows.

Definition 4.2.1 (Differential Privacy). *A randomized algorithm $M : \mathcal{D} \mapsto \mathbb{R}$ is (ϵ, δ) -DP, if for any pair of neighbouring datasets $D, D' \in \mathcal{D}$, and for any $S \subseteq \mathbb{R}$ we have*

$$\Pr[M(D) \in S] \leq e^\epsilon \Pr[M(D') \in S] + \delta. \quad (4.2)$$

The privacy parameter ϵ defines how similar the output distributions must be, and δ allows a small chance of failure in the definition. We use the unbounded neighbouring definition, where datasets are neighbours if $|D \setminus D' \cup D' \setminus D| = 1$ (we allow for the addition or removal of a single data point). Arbitrary computations can be carried out on the output of a DP mechanism without affecting privacy due to the post-processing lemma [70]. Finally, DP is composed naturally with multiple runs of a mechanism. If we apply differentially private mechanisms sequentially, the privacy parameter composes through summation or more advanced methods [70].

Definition 4.2.2 (Sensitivity). *Let $f : \mathcal{D} \mapsto \mathbb{R}^k$. If $\mathbb{D}$ is a distance metric between elements of $\mathbb{R}^k$, then the $\mathbb{D}$ -sensitivity of f is*

$$\Delta^{(f)} = \max_{(D, D')} \mathbb{D}(f(D), f(D')), \quad (4.3)$$

where (D, D') are pairs of neighbouring datasets.

We primarily use the ℓ_2 norm as the distance metric $\mathbb{D}$. To satisfy DP, we use Gaussian noise in this work and analyze it with Gaussian Differential Privacy (GDP) [61].

Definition 4.2.3 (GDP [61]). *A mechanism M is said to satisfy θ -Gaussian Differential Privacy (θ -GDP) if*

$$\mathcal{T}(M(D), M(D')) \geq G_\theta$$

for all neighbouring datasets D and D' , where $\mathcal{T}$ is a trade-off function measuring the difficulty for attackers in identifying the presence of an individual data point and $G_\theta = \mathcal{T}(\mathcal{N}(0, 1), \mathcal{N}(\theta, 1))$ (see Dong et al. [61] for specifics of the definition).

Naturally, the Gaussian Mechanism satisfies GDP.

Theorem 4.2.1. (Gaussian Mechanism for GDP [61]) Define the Gaussian mechanism that operates on a statistic f as $M(D) = f(D) + \eta$, where $\eta \sim \mathcal{N}(0, (\Delta^{(f)})/\theta)$. Then, M is θ -GDP.

We consider the composition of GDP over multiple runs of a mechanism.

Theorem 4.2.2 (GDP Composition [61]). The n -fold composition of θ_i -GDP mechanisms is $\sqrt{\theta_1^2 + \dots + \theta_n^2}$ -GDP.

We also have a way to convert between GDP and DP.

Algorithm 3 **PE-means** based on Private Evolution (PE) [135]

Input: Private samples: $D = \{x_i\}_{i=1}^N \subset \mathbb{R}^d$
 Number of iterations: T
 Number of generated samples: $N_{\text{syn}} = k$
 Number of initial variations: L
 Noise multiplier for the DP Nearest Neighbors Histogram: σ

Output: DP cluster centres $S'_T = \{\mu_1, \dots, \mu_k\} \subset \mathbb{R}^d$

```

1:  $S_0 \leftarrow \text{RANDOM\_API}(N_{\text{syn}} * (L + 1))$ 
2: for  $t \leftarrow 1, \dots, T$  do
3:    $\text{hist}_t \leftarrow \text{DP\_NN\_HISTOGRAM}(D, S_{t-1}, \sigma)$  ▷ Algorithm 4
4:    $S'_t \leftarrow \text{select top-}N_{\text{syn}} \text{ samples from } S_{t-1} \text{ ranked by } \text{hist}_t$ 
5:    $S'_t \leftarrow \text{WEIGHTED\_K\_MEANS}(S_{t-1}, k=N_{\text{syn}}, \text{weight}=\text{hist}_t)$ 
6:    $S_t \leftarrow S'_t$  ▷ Save best samples
7:   if  $\text{sum}(\text{hist}_t^2)/k(L+1)\sigma^2 < 1.0$  then
8:      $L \leftarrow L/2$ 
9:   for  $i \leftarrow 1, \dots, L$  do
10:     $S_t \leftarrow S_t \cup \text{VARIATION\_API}(S'_t)$ 
11: return  $S'_T$ 

```

Theorem 4.2.3 (GDP to DP [24,61]). A mechanism is θ -GDP if and only if it is $(\epsilon, \delta(\epsilon))$ -DP for all $\epsilon \geq 0$, where

$$\delta(\epsilon) = \Phi\left(-\frac{\epsilon}{\theta} + \frac{\theta}{2}\right) - e^\epsilon \Phi\left(-\frac{\epsilon}{\theta} - \frac{\theta}{2}\right).$$

In practice, we follow the implementation of PE¹ to solve this function for θ , given a desired ϵ and δ .

4.3 Methodology

Private Evolution (PE) is an emerging private synthetic data generation technique that evolves a set of synthetic samples using only inference API access to large machine learning models [134, 135, 193, 201]. At a high level, PE starts from randomly generated synthetic

¹<https://github.com/microsoft/DPSDA>

data (using the `RANDOM_API`) and iteratively evolves toward high-quality synthetic data by creating new variations (using the `VARIATION_API`) and ranking the synthetic data that is closest to the private data. We build upon the text variant of PE called Aug-PE [201]. In Algorithm 3, we outline the baseline version of PE. Algorithm 3 also illustrates our approach, PE-means, highlighting our modifications to both improve PE and adapt it to solve k -means clustering. In this section, we detail the design of PE-means with respect to each part of the algorithm. Following this, we present our high-dimensional variant of PE-means in Section 4.3.5 and prove the privacy of both algorithms in Section 4.3.6.

4.3.1 Random API

The first step in PE is to generate an initial population, independently of the private data, using the `RANDOM_API` (Line 1). In the earliest variants of PE [135, 201], this is done by querying a pre-trained foundation model (using information such as the label, which is assumed to be public). To solve clustering with PE, rather than creating a population of synthetic data samples, we create a population of centroids to evolve. We assume the only context available to generate the initial population is the shape of the data domain. For simplicity, we assume the domain is a hyper-sphere and use the same ℓ_2 bound R on its radius that other approaches need to bound sensitivity [21, 39, 55, 186]. However, in our case, the bound being too tight or loose can at most affect convergence speed and has no effect on the amount of noise added. A naive `RANDOM_API` for clustering would simply generate uniformly random data within the hyper-sphere. However, random points may group together, which reduces our coverage of the domain. Thus, we instead follow the work of Su et al. [186] and use the sphere packing method. Su et al.’s method starts with a given radius a (typically half the radius of the domain), and iteratively adds random samples that are at least a away from the boundary and $2a$ from the samples added so far. If, after a fixed number of retries, a random sample cannot be found to meet these conditions, a is halved and the algorithm continues until k samples are found. We adapt Su et al.’s algorithm to consider an ℓ_2 bounded hyper-sphere rather than an ℓ_∞ bounded hypercube. The rest of the algorithm remains the same.

4.3.2 Selection Step

The next step in PE is to select the best candidates from the population (either from the random initialization or previous iteration). First, the selection computes a private histogram of votes, then selects candidates based on these votes. We detail our modifications to both parts.

Algorithm 4 **Modified** DP_NN_HISTOGRAM

Input: Private samples: D

Generated samples: $S = \{z_i\}_{i=1}^n$

Noise multiplier: σ

Distance function: $d(\cdot, \cdot)$

Output: DP nearest neighbours histogram on S

```
1:  $histogram \leftarrow [0, \dots, 0]$ 
2: for  $x_{priv} \in D$  do
3:    $i = \arg \min_{j \in [n]} d(x_{priv}, z_j)$ 
4:    $histogram[i] \leftarrow histogram[i] + 1$ 
5:  $histogram \leftarrow histogram + \mathcal{N}(0, \sigma I_n)$ 
6:  $histogram \leftarrow \text{sort\_descending}(histogram)$ 
7:  $k \leftarrow \min \left\{ \kappa \mid \sum_{j=1}^{\kappa} histogram[j] > N \right\}$ 
8:  $histogram[j] \leftarrow 0$  for all  $j > k$  ▷ Zero out noisy values
9: return  $histogram$ 
```

Histogram Computation To utilize the private data in the selection, we compute the DP_NN_HISTOGRAM (Line 3) described in Algorithm 4. The idea is that all private samples vote for their nearest neighbour in the population (Lines 3 and 4). This creates a histogram of votes that is easy to privatize using Gaussian noise in Line 5. However, depending on the size of the population and progression of the evolution, this histogram can be quite sparse. Some of the previous PE versions [134, 135, 193] post-processed the histogram by zeroing out any noisy votes below a certain threshold. The challenge is how to choose the optimal threshold without spending privacy budget. We propose a modification based on the constraint that, before adding noise, the histogram of votes should be non-negative and sum to the size of the dataset (which we assume to be public). The maximum likelihood estimation (MLE) of the true votes given these constraints is found by projecting the noisy votes onto the set $\{x \in \mathbb{R} \mid x_i \geq 0, \sum_i x_i = N\}$. Duchi et al. [65] give an efficient algorithm for this projection by computing a subset of bins with the largest noisy counts such that after scaling and thresholding they sum to exactly N . For clustering, we do not need the sum constraint to be exact; a more critical issue is points far from the private data getting assigned a few false votes due to noise and pulling the centroids in the wrong direction. Thus, we give a simple approximation of the MLE algorithm that computes the minimum number of highest-count buckets that exceed the sum constraint (Line 7) and zeros out the remaining buckets (Line 8), without any additional scaling. Intuitively, we

assume histogram buckets with large counts are likely to have dominated the noise.

Selecting Candidates After computing the private histogram, the next evolutionary task is to choose the best candidates from the population to survive to the next iteration based on this histogram. Aug-PE takes the approach of ranking candidates based on the number of votes they received in the histogram (Line 4). The drawback to this approach is that choosing the most popular points does not always yield optimal clustering coverage due to what we call vote splitting. Consider a scenario where PE generates a point that is the nearest neighbour of many private dataset points. In the next iteration, PE may generate numerous new samples close to this point, causing the votes to be split between these new samples. Then, since all new points have a low count, an entire area of feature space may not be selected. We illustrate an example of this for clustering in Section 4.4.2. We argue that instead of just choosing points with the highest votes, we also need to choose one or two solutions from areas with highly split votes. Our key observation is that standard k -means clustering algorithms have a similar objective. Thus, we use the population as a coreset weighted by the histogram and run standard k -means, using the resulting centroids as the selected set for the next round (Line 5).² We show that this technique is much more robust to vote splitting in Section 4.4.2.

4.3.3 Variation API

After selecting the best candidates, the next step is to continue exploring the space for better centroids by creating variations of the selected candidates. We note that we preserve the best candidates (referred to as elitism in evolutionary algorithms) in Line 6, so that if no variation improves, we maintain the current progress. In some of the previous versions of PE, the `VARIATION_API` (Line 10) would be another API call to a foundation model. For example, Aug-PE [201] would randomly mask words and ask a large language model to fill them in with a high temperature. In the case of clustering, our `VARIATION_API` perturbs the real-valued candidate centroids directly. We find that mutation based on Lévy flights works well, as the distribution is heavier-tailed to encourage exploration. We follow Mantegna’s algorithm [146] to generate a Lévy stable distribution parameterized by the Lévy index β which we denote by $\text{Lévy}(\beta)$. To sample a value $\zeta \sim \text{Lévy}(\beta)$, we first

²We could instead use a k -medians algorithm to select candidates strictly from the population satisfying a similar objective. However, we see no reason to enforce that constraint in this work, as the standard k -means algorithm can move us closer to the true centroids while making the selection.

compute

$$\sigma_u = \left(\frac{\Gamma(1 + \beta) \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \cdot \beta \cdot 2^{(\beta-1)/2}} \right)^{1/\beta}, \quad (4.4)$$

then draw independent samples $u \sim \mathcal{N}(0, \sigma_u I_d)$ and $v \sim \mathcal{N}(0, I_d)$ that we combine to get $\zeta = \frac{u}{|v|^{1/\beta}}$. Our variation API adds i.i.d. noise from this distribution to the existing centroids, scaled by a hyperparameter γ and the radius R , namely

$$\text{VARIATION\_API}(\mu_i) = \mu_i + \gamma R \zeta \quad (4.5)$$

where $\zeta \sim \text{Lévy}(\beta)$. Finally, we clip any mutated sample with ℓ_2 norm greater than R to the boundary. We experimentally set the hyperparameters for our variation API and validate the use of Lévy over Gaussian noise in Section 4.4.3.

4.3.4 Adaptive Population Size

Following Aug-PE, we apply the variation API L times to each sample to increase exploration (Line 10). Tuning the L parameter is critical. If L is too small, the algorithm converges slowly, needing more iterations and privacy budget. Alternatively, if L is too large, the histogram becomes sparse, resulting in a poor signal-to-noise ratio and poor-quality selections. While we can tune L as a hyperparameter (we show a heuristic for setting it in Section 4.4.3), the best setting for L varies greatly depending on the distribution of the private data and the current distribution of the candidates. To help avoid the more damaging case of a large L preventing meaningful selection, we add adaptive tuning of L during evolution. In Line 7, we estimate the signal-to-noise ratio (without using any privacy budget) by computing the ℓ_2 norm of the noisy histogram vs. the expected ℓ_2 norm of the noise. We argue that if this value is less than 1, there is likely more noise than signal, and we reduce L by half (Line 8). We experimented with different thresholds and amounts of reduction but found these settings work well in practice.

4.3.5 High-Dimensional Case

In high-dimensional domains (especially when k is small), variations become less effective due to the curse of dimensionality. To combat this, in Algorithm 5, we present a modified version of PE-means that is optimized for higher dimensions. We follow the work of Balcan et al. [21] and first use a Johnson-Lindenstrauss transform to project the private data to a

Algorithm 5 High-Dimensional PE-means (HDPE-means)

Input: Private samples: $D = \{x_i\}_{i=1}^N \subset \mathbb{R}^d$

Reduced dimension: p

Number of clusters: k

Number of PE iterations: T

Number of PE variations: L

Noise multiplier: σ

Domain radius: R

Output: DP cluster centres $S = \{\mu_1, \dots, \mu_k\} \subset \mathbb{R}^d$

- 1: Sample random projection matrix $G \sim \mathcal{N}(0, 1)^{p \times d}$
 - 2: $\bar{D} \leftarrow \{\bar{x}_i\}_{i=1}^N$ where $\bar{x}_i = \frac{1}{\sqrt{d}} G x_i$ ▷ Project data to $\mathbb{R}^p$
 - 3: $\{\bar{\mu}_1, \dots, \bar{\mu}_k\} \leftarrow \text{PE-means}(\bar{D}, T - 2, k, L, \sigma)$ ▷ Get centroids in $\mathbb{R}^p$ using Algorithm 3
 - 4: **for** $i \leftarrow 1, \dots, N$ **do**
 - 5: $a_i \leftarrow \arg \min_{j \in \{1, \dots, k\}} \|\bar{x}_i - \bar{\mu}_j\|_2$ ▷ Assign labels in $\mathbb{R}^d$
 - 6: **for** $j \leftarrow 1, \dots, k$ **do**
 - 7: $z_j \leftarrow \sum_{i: a_i=j} x_i + \mathcal{N}(0, R\sigma I_n)$ ▷ Noisy sum in $\mathbb{R}^d$
 - 8: $n_j \leftarrow \sum_{i: a_i=j} 1 + \mathcal{N}(0, \sigma I_n)$ ▷ Noisy count
 - 9: $\mu_j \leftarrow \frac{z_j}{n_j}$
 - 10: **return** $S = \{\mu_1, \dots, \mu_k\}$
-

low-dimensional space (Line 2). We then run PE-means as normal in this low-dimensional space in Line 3. To project the resulting cluster centres back to the original domain, we use noisy averaging. This can be thought of as a single iteration of DP Lloyd’s algorithm [186] in the original domain, initialized with the low-dimensional cluster assignments. Specifically, all private data points are assigned a label using the low-dimensional centroids (Lines 4 to 5). Then we compute the sum and count of all private points for a given label to obtain the centroids in high-dimensional space (Lines 6 to 9). To ensure privacy, we add Gaussian noise to the sum and count of points in each cluster. We reduce the number of iterations for PE by two in Line 3, so we can use the same noise multiplier to noise the sum and count. We show in Section 2.6 that HDPE-means outperforms PE-means in higher dimensions. The disadvantage of this approach is that, similar to related work [21, 39], we add noise proportional to R (the radius of the data domain) during the projection back to high dimensions. However, most other approaches apply strictly more noise, by applying radius-proportional noise in each iteration. Our approach takes advantage of the

low sensitivity of PE-means to fit the centroids accurately in low-dimensional space before conducting a single iteration of radius-proportional noise in the final iteration.³ In future work, we will investigate alternate ways to scale the variation API while reducing this dependency.

4.3.6 Privacy Analysis

The privacy analysis of PE-means follows from the original PE paper [135] as our changes do not affect noise addition in Algorithm 4 or use private data in any way. We restate the analysis here for completeness.

Theorem 4.3.1. PE-means as defined in Algorithm 3 satisfies (ϵ, δ) -DP.

Proof. We break the proof into the same steps as the original paper [135]:

- **Step 1: Bounding the sensitivity (Definition 4.2.2) of the DP Nearest Neighbors Histogram in Algorithm 4.** Each private sample only contributes one vote. If we add or remove one sample, the resulting histogram will change by 1 in the ℓ_2 norm. Therefore, the sensitivity is 1.
- **Step 2: Viewing each PE iteration as a Gaussian mechanism.** In Line 5 of Algorithm 4, we add i.i.d. Gaussian noise with standard deviation σ to each bin. This is the only part of the algorithm that touches the private dataset.
- **Step 3: Viewing the entire PE algorithm as T adaptive compositions of Gaussian mechanisms.** This holds because PE simply applies algorithm 4 T times sequentially.
- **Step 4: Viewing the entire PE algorithm as one Gaussian mechanism with noise multiplier $\sigma/\sqrt{T}$.** Using the composition theorem (Theorem 4.2.2) from Dong et al. [61, Corollary 2], we get that T applications of a $1/\sigma$ -GDP algorithm are $\sqrt{T}/\sigma$ -GDP.
- **Step 5: Computing DP parameters ϵ and δ .** The problem is simply computing a σ such that (ϵ, δ) -DP iff $\sqrt{T}/\sigma$ -GDP, for which we apply the formula of Balle and Wang [24].

³We note that Google-LSH also only uses a single iteration of radius-proportional noise. However, their approach assigns cluster membership based on a single LSH partition of the space, whereas we iteratively optimize cluster membership with PE, yielding a more accurate clustering.

□

Our high-dimensional extension of PE includes additional use of the private data in the projection, and thus we state and prove its privacy.

Theorem 4.3.2. HDPE-means as defined in Algorithm 5 satisfies (ϵ, δ) -DP.

Proof.

- **Step 1: Composition accounting.** We first assume that PE is called (in Line 3) with two fewer iterations than the number of iterations σ was computed for. Applying Theorem 4.3.1, we get that the run of PE is private with two extra applications of a Gaussian mechanism with standard deviation of σ remaining.
- **Step 2: Assignment as post-processing.** After calling PE, the assignment step (Lines 4 to 5) uses the published centroids from the previous iteration (post-processing) to divide the dataset into clusters. We can then apply parallel composition over each of the clusters. Thus, we can focus on the privacy cost of a single cluster for the remainder of the proof.
- **Step 3: Privacy of the Gaussian mechanisms.** For each cluster, we apply the Gaussian mechanism twice. The first is to compute a sum which has an ℓ_2 sensitivity of R since adding or removing a point can at most add the largest possible vector in the domain. The second is to compute the count, which has an ℓ_2 sensitivity of 1, as a data point can be counted at most once. Applying Theorem 4.2.1, the result follows.

□

4.4 Experiments

In this section, we provide a complete experimental evaluation of PE-means and HDPE-means. We begin by detailing the experimental setup in Section 4.4.1, followed by a case study highlighting the effect of vote splitting in practice in Section 4.4.2. Section 4.4.3 chooses and validates effective default hyperparameters for our approach. Finally, Section 4.4.4 gives a complete utility benchmark compared to the baselines with additional discussion in Section 4.4.5.

4.4.1 Experimental Setup

Datasets We use a combination of real and synthetic datasets from various sources. Most of the real datasets as well as the G2 (Gaussian-based synthetic) datasets come from the clustering datasets repository [81], with some additional real datasets from the UCI machine learning repository [64]. For Birch2 [208], we take 25,000 random samples from the dataset of 100,000. Gas [64], Letter [64], and MNIST [127] are included for comparability with Chang and Kamath [39]. For MNIST, we train a LeNet5 model and use neural representations following Chang and Kamath [39]. The *scale* synthetic datasets from Diaa et al. [55] were generated with the `clusterGeneration` R package [172] and include a random level of cluster overlap. Finally, we include synthetic datasets generated with the `make_blobs` function from Sklearn that generates isotropic Gaussian clusters. Table 4.1 includes a complete list of datasets and their sizes.

Baselines Our baselines are selected from state-of-the-art DP k -means algorithms with publicly available implementations. The work of Chang and Kamath (which we denote by *Google-LSH*) is perhaps the most popular approach [39]. Chang and Kamath use a locality-sensitive hash tree to create a private coreset, upon which the non-private k -means algorithm is applied. We modify their code to fit our assumption that the size of the dataset is public (requiring less noise). Because the Google-LSH method is one of the most recent works, we also evaluate the approaches from its evaluation. The first is IBM’s differential privacy library [99] (which we denote by *DP-Lib*) that implements an improved version of Su et al.’s algorithm [186]. Su et al. apply noise to each step of Lloyd’s clustering algorithm [139] with various optimizations such as the sphere packing initialization. The second is the work of Balcan et al. (which we denote by *Icml17*) that first projects the data to a low-dimensional space before recursively dividing the space into cubes. They then apply a k -medians-style swapping algorithm to choose the best centres before projecting the result back into the high-dimensional space through noisy averaging. We discovered several inconsistencies in the privacy analysis of Balcan et al.’s implementation, which we detail in the documentation of our implementation. We note that both DP-Lib and Icml17 work in the pure differential privacy model and thus, following Google-LSH [39], we evaluate them with $\delta = 0$. The final related work is a recent improvement over Su et al.’s algorithm by Diaa et al. [55], which we denote by *FastLloyd*. FastLloyd modifies Su et al.’s work by using Gaussian noise and computing cluster updates relative to the previous iteration, thereby reducing the sensitivity.

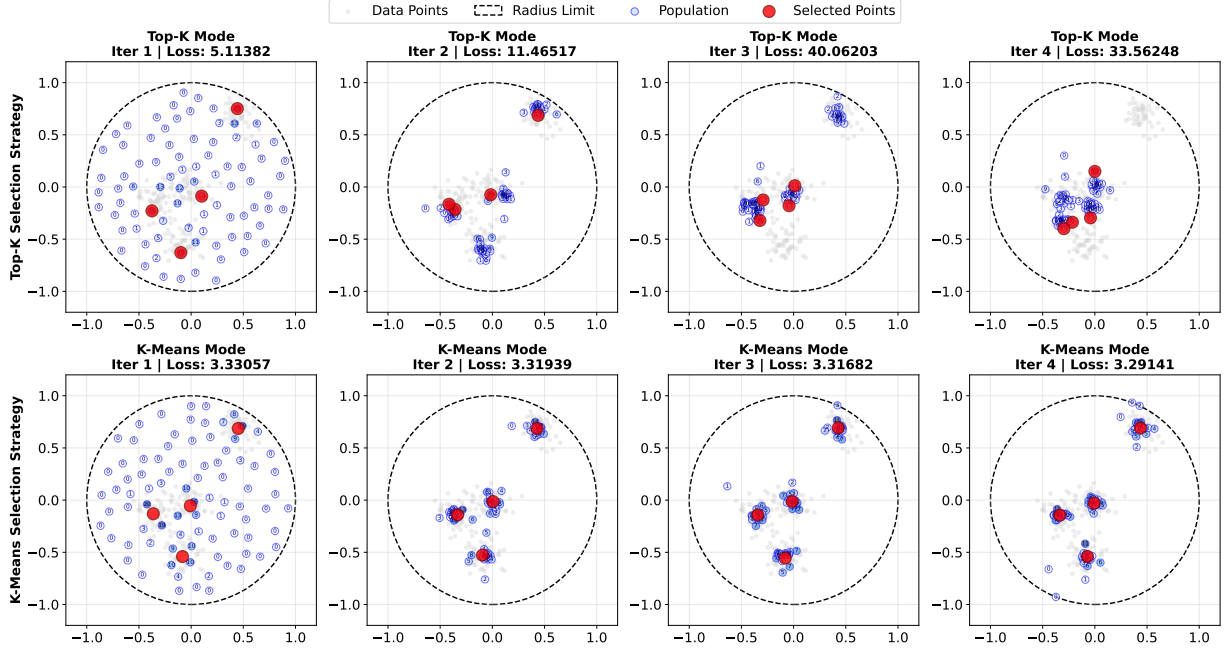


Figure 4.1: A visualization of the private data, samples generated, and samples selected by PE in each iteration, comparing the previous top- k selection approach to our proposed k -means approach with $\epsilon = \infty$. The k -means approach effectively mitigates the vote splitting problem and keeps selected points close to the true clusters.

Implementation Details Our source code can be found at <https://github.com/t3humpries/PE-means>. We normalize all datasets to have a max ℓ_2 norm of 1, for a clean presentation of metrics over different datasets. We follow Google-LSH and centre each dataset first by subtracting the mean [39]. Then, also following Google-LSH, we non-privately compute the ℓ_2 and ℓ_∞ sensitivity bounds on each dataset. In practice, this bound should be computed privately or derived from public information about the attributes. However, for the sake of comparison, we give all approaches an equal advantage by providing a tight bound on the domain. Each experiment is repeated 50 times over different random seeds, and we report the average. All shaded areas represent the 95% confidence interval of the mean of the results. We fix the privacy parameter $\delta = 1/N^{1.1}$, following the best practice that the failure probability should be less than $1/N$ [70]. The primary loss we consider is the normalized k -means loss.

$$\text{Loss} = \frac{1}{N} \sum_{i=1}^N \min_{j=1}^k \|x_i - \mu_j\|^2 \quad (4.6)$$

We also compute each method’s performance over different privacy budgets using the area under the curve (AUC) of the loss values against $\epsilon \in \{0.25, 0.5, 1.0, 2.0, 4.0\}$, computed via the trapezoidal rule:

$$\text{AUC} = \sum_{i=1}^{n-1} \frac{\text{Loss}_i + \text{Loss}_{i+1}}{2} \cdot (\epsilon_{i+1} - \epsilon_i) \quad (4.7)$$

4.4.2 Vote Splitting Example

In Section 4.3.2, we described the issue of vote splitting in PE’s selection. In this section, we illustrate the phenomenon with a toy example. We generate a two-dimensional random dataset with $k = 4$ using the `make_blobs` function and set $\epsilon = \infty$. In Figure 4.1, we plot four iterations of PE (one per column) and show the difference between the previous top- k voting (top row of plots) and our weighted k -means selection (bottom row of plots). The plots include the private data in grey, the population of PE in blue, and the selected points for each method in red.

In the case of the top- k voting (first row), we observe that the initial iteration chooses points close to the four true clusters in the dataset. However, in the second iteration, there are many good candidates in the lower cluster, splitting the vote and causing all selected points to be in the other three clusters. Then, in the third iteration, the lower cluster dies out and the vote becomes split in the upper cluster. The clusters do not recover in the fourth iteration, as it will now take PE numerous iterations to move back towards those areas. In contrast, the bottom row of plots shows PE maintaining and refining four healthy clusters throughout the iterations. We note that in this toy example, PE has essentially converged in the first iteration, which further highlights the challenge with top- k voting.

4.4.3 Hyperparameter Tuning

Iterations and Variation Size

Two of the most influential hyperparameters for PE-means are the number of iterations T and the number of variations L . For a fixed privacy budget, if the number of iterations is too high, the amount of noise used in each iteration grows too large, preventing meaningful selections. If the algorithm executes too few iterations, it will not converge. Similarly, if L is too large, the histogram can become sparse and the votes will not dominate even the smallest amount of noise. If L is too small, then the chance of PE-means finding points closer to the private data is reduced, slowing convergence. We recall that our adaptive

reduction of L (Line 8 of Algorithm 3) protects against the case when L is causing the noise to dominate the signal, but it is still beneficial to set a good starting point.

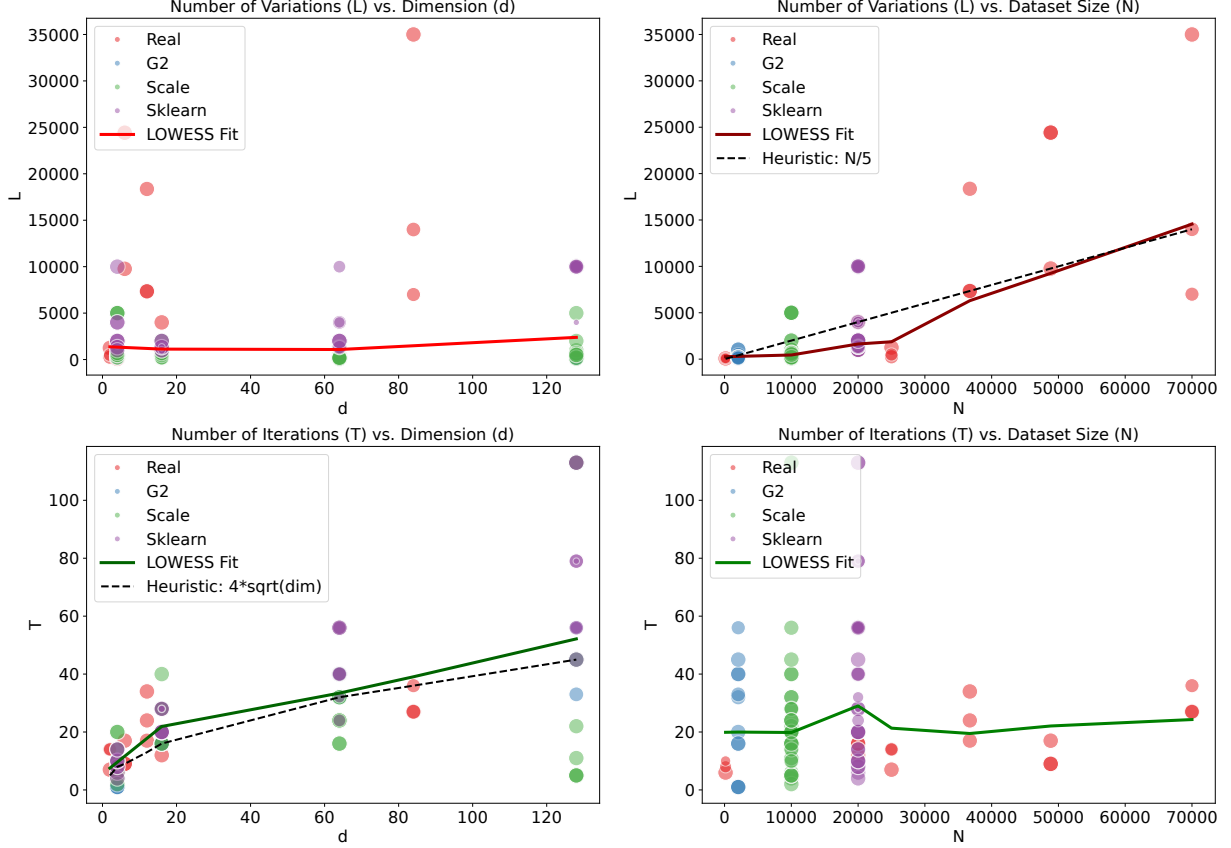


Figure 4.2: A scatter plot of the top-3 parameter configurations of L and T for each dataset at $\epsilon = 1$. The LOWESS line of best fit shows a clear relationship between L and N and between T and d that are captured by our heuristics.

The only public parameters we can use to estimate the difficulty of a dataset in the private setting are its dimensions. In our initial testing, we observed that the number of iterations is influenced by the dimension of the dataset. Namely, in higher-dimensional space, PE needs more steps due to the curse of dimensionality. We also observed that the sparsity of the histogram for a given L is highly influenced by the number of values in a dataset that can vote. To investigate this more rigorously, we conduct an experiment where we vary L and T for each dataset and plot the values against the dataset size and dimension. This experiment uses all datasets to give a variety of datasets spanning dimensions, dataset

sizes, and values of k . We plot the results in Figure 4.2, where $\epsilon = 1$. We scatter the values of the top-3 best-performing parameters for each dataset and draw a line of best fit using the LOWESS method. While the top-left plot reveals no useful relationship between the number of variations L and the number of dimensions d , the top-right plot shows a clear relationship between the dataset size N and L . We approximate this relationship with the heuristic $L = \max(N/5, 4)$. Similarly, we see a clear relationship between the dimension d and number of iterations T in the bottom-left plot, which we approximate with the heuristic $T = \max(4\sqrt{d}, 1)$. Finally, we see no clear relationship between N and T in the bottom-right plot. In our experiments, we never encounter the constant cases in the max terms, but include them to ensure a robust implementation.

While Figure 4.2 demonstrates that our heuristics closely follow the line of best fit, it is also clear that the optimal settings for a number of datasets are quite far away from the line. To further investigate this, we consider the difference in utility between the optimal settings and the heuristic settings in Figure 4.3. The x-axis represents the loss of the optimal configuration of T and L over the entire grid search, and the y-axis represents the loss when T and L are set following the heuristic. We can see most points lie on or very close to the line $y = x$, indicating near-optimality. Thus, we argue our heuristics offer good default values in practice.

To extend these heuristics to also account for the privacy budget ϵ , we experimented with scaling them by factors such as $\epsilon, 0.5\epsilon, 2\epsilon, \sqrt{\epsilon}$ (since both parameters should decrease with smaller epsilon). We find that for values of $\epsilon < 1$, no modification improves over the baseline, as our adaptive strategy increases the signal-to-noise ratio for any dataset that needs it (and not all do). However, when $\epsilon > 1$, we find it is best (taking the median performance over all datasets) to scale the number of iterations by ϵ ($T = \max(4\epsilon\sqrt{d}, 1)$).

Variation Parameters

The remaining hyperparameters are those of our variation API. The first is the Lévy index β that determines how heavy the tails of the distribution are. Smaller values indicate a heavy tail with $\beta = 1$ representing a Cauchy distribution, and $\beta = 2$ representing a Gaussian. In our implementation, we add a condition to explicitly draw from a standard Gaussian at $\beta = 2$. The second parameter, the variation scale γ , is a multiplicative factor to control how much to vary the centroids. We conduct a grid search over possible values of each parameter (with L and T fixed to the heuristics from the previous section) and plot the results in Figure 4.4. We plot the grid search results as a heatmap with each cell showing the percentage increase of this parameter configuration over the optimal parameter configuration, averaged over all datasets we evaluate (see Table 4.1 for the complete list of

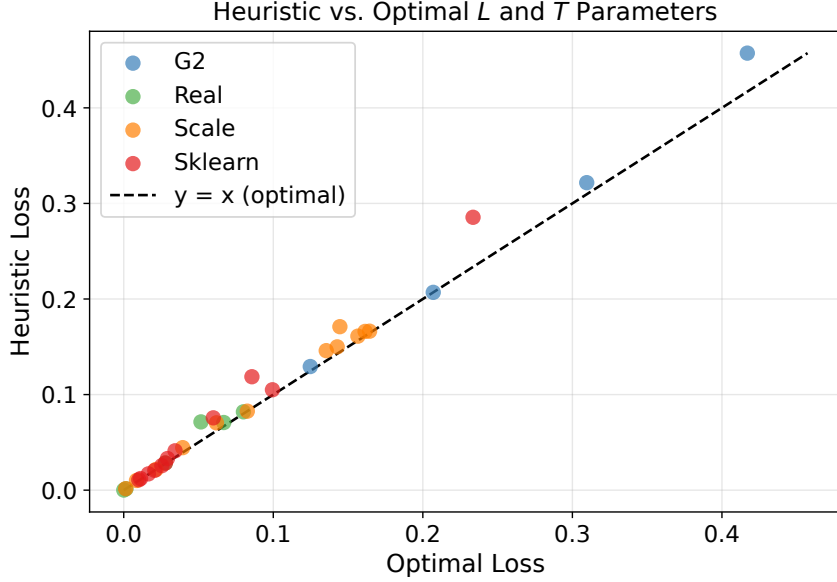


Figure 4.3: A scatter plot of grid search results over L and T with $\epsilon = 1$. There is one point per dataset with the x-axis shows the clustering loss of the optimal parameter setting in the grid and the y-axis showing the loss of our heuristic for that dataset. We observe that the loss of our heuristic parameter settings are close to the optimal loss for all datasets.

datasets). We can see that $\beta = 1.75$ and $\gamma = 0.01$ give the best results and thus, we choose these values as our default. In Figure 4.5, we scatter the optimal vs. default values of β and γ over all datasets. We can see that the default parameters stay close to the optimal line of $y = x$, indicating a good choice of parameters.

4.4.4 Utility Benchmark

Overall Comparison We compare PE-means and HDPE-means to the baselines over all datasets and summarize the results in Table 4.1. The metric we use is the AUC of the k -means loss (a summary of performance over a collection of epsilons, with smaller values being best).⁴ We highlight the approach with the lowest loss AUC in green. HDPE-means is only included when the dimension of the dataset is greater than 16, as that is the dimensionality we project down to. Overall, PE-means and HDPE-means perform the best

⁴To convert the non-private k -means loss to an AUC score, we compute the area under the horizontal line drawn at the non-private utility over the same set of epsilons evaluated for the private approaches.

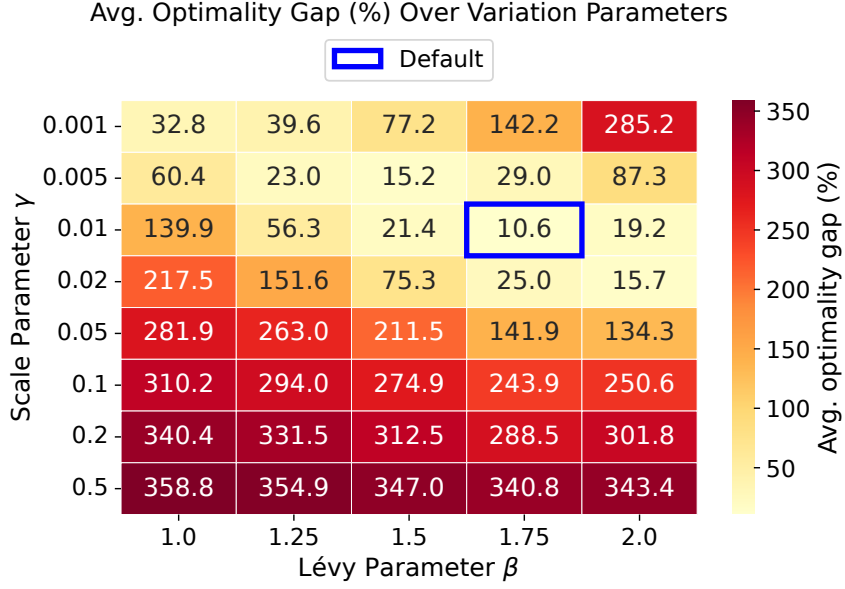


Figure 4.4: A heat map of the grid search results over γ and β at $\epsilon = 1$ with each cell showing the percentage increase in clustering loss for a given hyperparameters setting compared to the optimal hyperparameter choice for each dataset, averaged over all datasets. We observe our default parameter choice is the closest to the optimal loss.

over most datasets, with FastLloyd doing well on the g2_4 dataset, and Google-LSH doing well in two of the high-dimensional scale datasets. We give the percentage of improvement (or decline) of the best PE-based approach compared to the best non-PE private approach in the last column. We see that when PE-based approaches are outperformed, it is by at most 2%, but on average, they improve by 26%, with improvements of as much as 91%. Finally, we note that HDPE-means typically outperforms PE-means on datasets with dimension larger than 16. In high-dimensional cases where PE-means outperforms HDPE-means, it is typically not by a significant amount compared to the performance of related work, and thus we recommend always using HDPE-means in $d > 16$ datasets.

Real Dataset Privacy vs. Utility Trade-off We give the full privacy vs. utility trade-off for the various real datasets (including the three evaluated by Google-LSH [39]) in Figure 4.6. We observe that a PE-based approach is always the best approach, being the closest to the non-private baseline in all plots. Among the baselines, we observe that FastLloyd performs well in the lower dimensions (birch2 and iris) and Google-LSH does

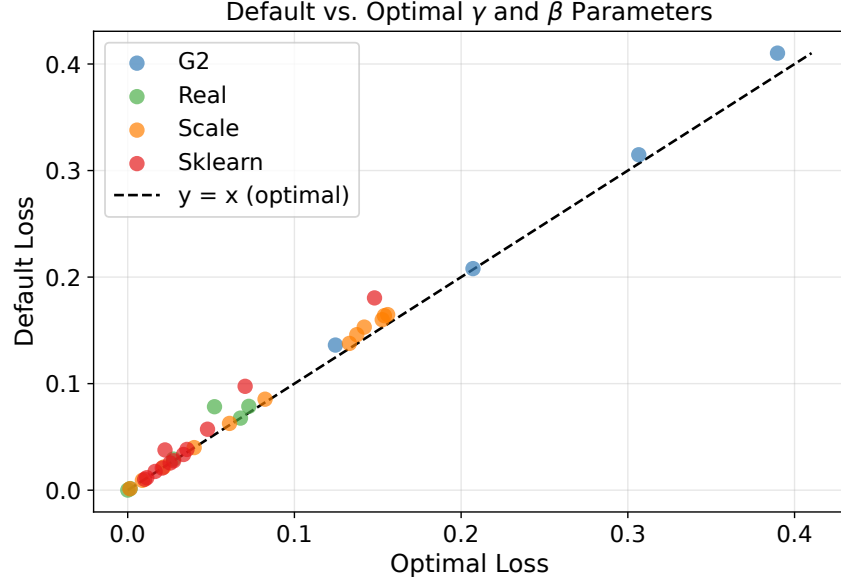


Figure 4.5: A scatter plot of grid search results over γ and β with $\epsilon = 1$. There is one point per dataset with the x-axis shows the clustering loss of the optimal parameter setting in the grid and the y-axis showing the loss of our default parameter values. We observe that the loss of our default parameter settings are close to the optimal loss for all datasets.

well in higher dimensions. We also note that our PE-based approaches scale well to limited privacy budgets.

Ground Truth Evaluation Following Google-LSH [39], we also evaluate an alternate metric called cluster label accuracy. We only evaluate this metric for datasets for which we have a ground-truth labelling, namely: mnist, letter, iris, birch2, and the G2 datasets. Given a clustering of the private data, we compute the metric by first assigning each centroid a label through a majority vote of the ground-truth label of the points in that cluster. Then we simply compute accuracy based on the number of private points correctly labelled by the centroid label. The results are given in Figure 4.7. We observe that for the letter, iris, and birch2 datasets, PE-means outperforms related work by a statistically significant amount as the confidence intervals do not overlap. In the remaining datasets, PE-means or HDPE-means perform similarly to the best baselines as well as the non-private baseline, indicating near-optimal performance.

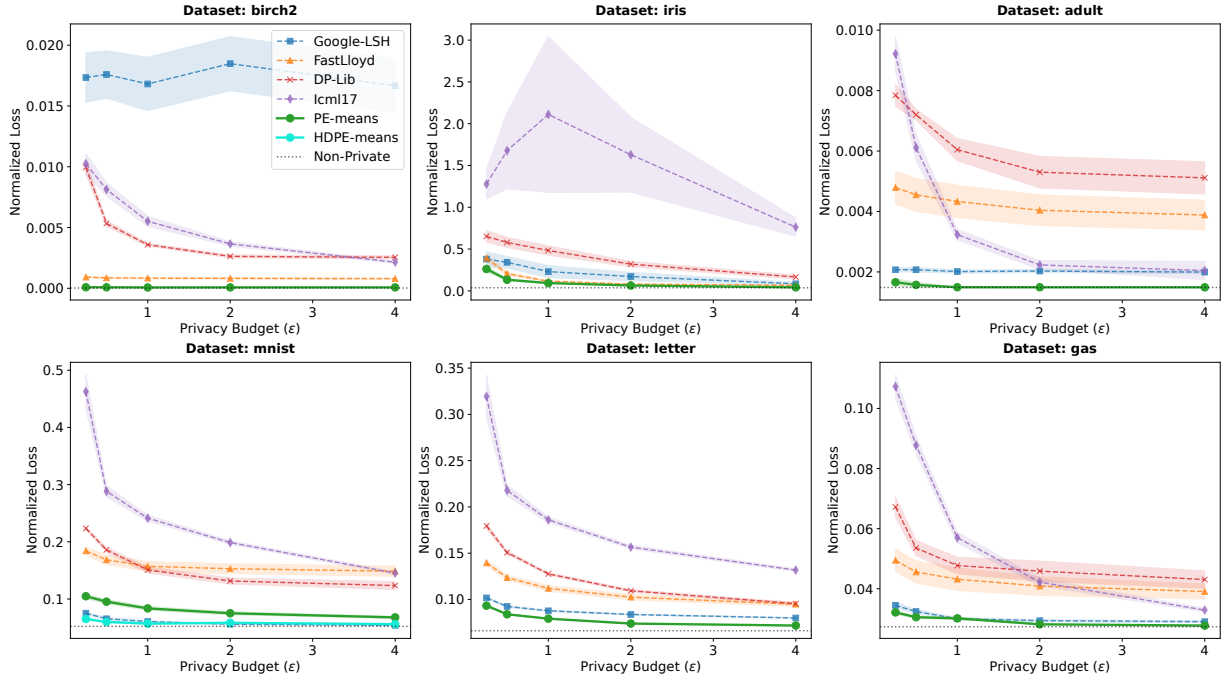


Figure 4.6: A plot of the privacy-utility trade-off of all approaches on real datasets, showing a PE-based approach always performs best.

Scaling Evaluation In Figure 4.8, we show how the algorithms scale with the number of clusters and a fixed privacy budget of $\epsilon = 1$. We omit Icml17 from the scale dataset plots for clarity as the loss is an order of magnitude worse than other approaches. The scale datasets follow the trend of the non-private version until we increase dimensions, at which point the noise has more of an effect at higher numbers of clusters. This is likely because the data size remains fixed, causing the number of samples voting (or being aggregated) per cluster to decline. The scale datasets also highlight the room for improvement in PE-based approaches in higher dimensions. For the Sklearn sets, which tend to be more well-separated Gaussian blobs, we observe PE-based approaches outperforming all other approaches and also scaling better with k .

4.4.5 Discussion

In addition to showing the improvements of PE-based approaches in clustering, our experimental evaluations also provide support for the PE algorithm in general. Specifically,

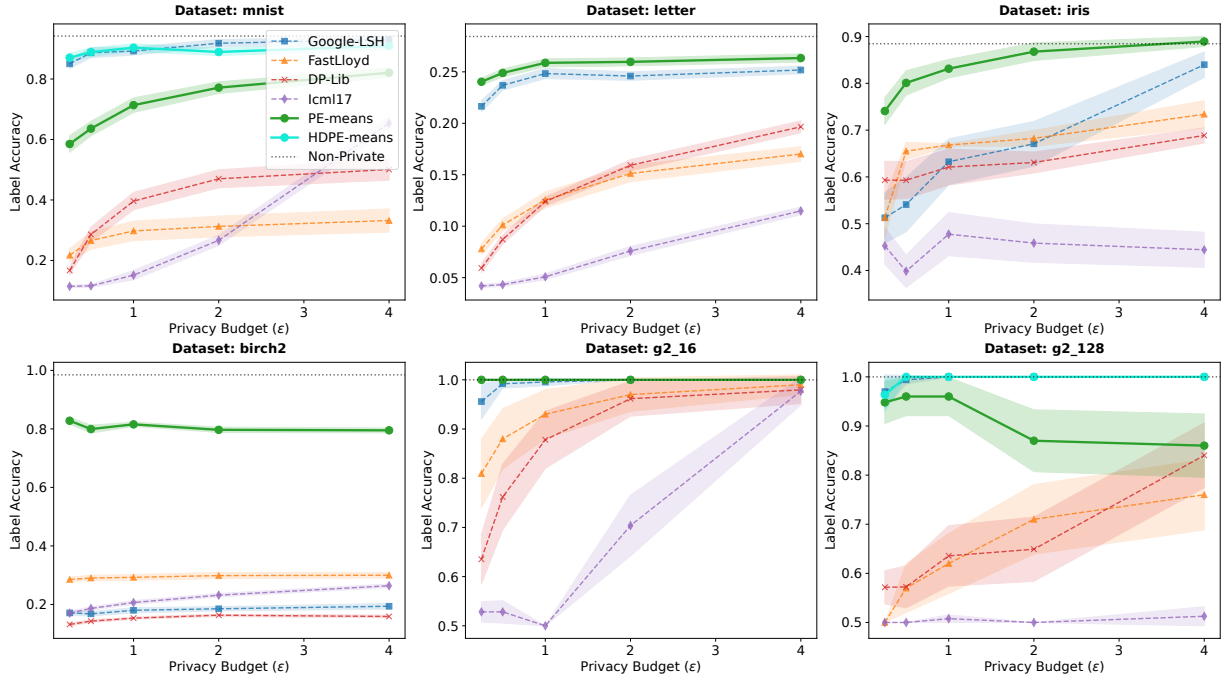


Figure 4.7: A plot of the label accuracy metric vs. privacy budget ϵ over all approaches on a subset of real datasets that have ground-truth labels. We observe that a PE-based approach always performs better or similar to state-of-the-art.

PE-means does not use foundation models to implement the random and variation APIs. This means we can rule out the effect of dataset contamination (evaluation datasets appearing in the foundation model’s training set) in our results. This gives additional evidence to support the effectiveness of the PE paradigm independently of the improvement of foundation models.

4.5 Related Work

DP k -means In addition to the baselines we compare against in Section 2.6, there are several previous works that also solve the DP k -means clustering problem, which are either improved upon by our baselines or are more theoretical in nature. One line of work focused on developing a private version of Lloyd’s algorithm [32, 55, 66, 150, 186] starting with Blum et al. [32] and concluding with our baselines of Su et al. [186] and FastLloyd [55]. Another line of work used the sample and aggregate framework [153, 166] to solve k -means, but was

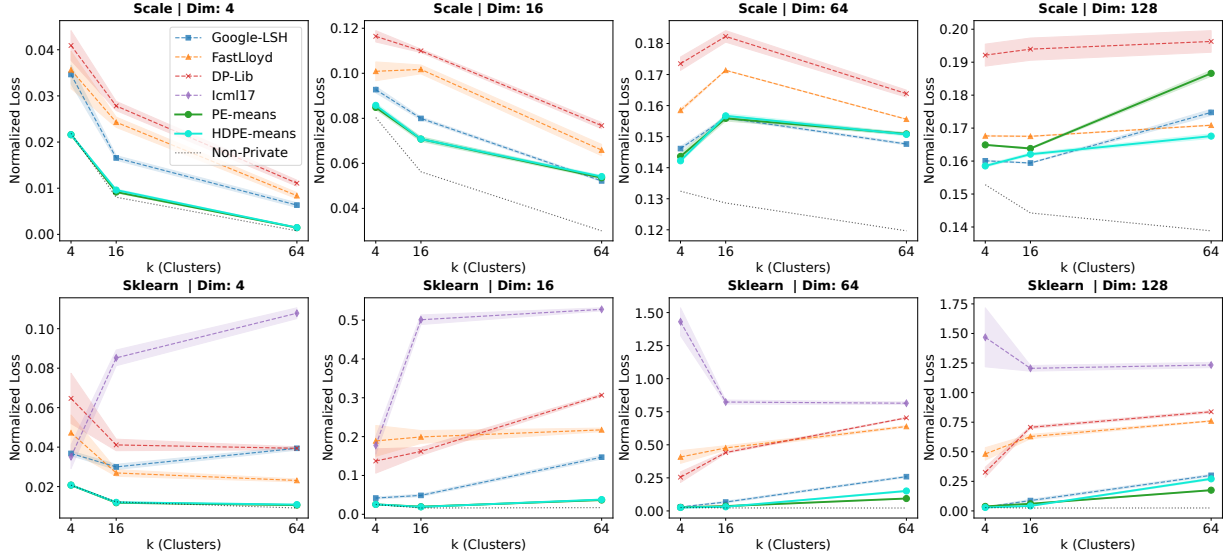


Figure 4.8: A comparison illustrating how approaches scale with the number of clusters k over different synthetic datasets at $\epsilon=1.0$.

outperformed by Su et al. [186]. A theoretical line of work focused on minimizing the bounds on approximation error, but did not provide experimental evaluation [79, 80, 84, 114, 167, 185].

DP Evolutionary Algorithms There has been previous work that has applied evolutionary techniques to the problem of clustering. The first differentially private evolutionary algorithm was developed by Zhang et al. [207], who evaluated it on the k -means clustering problem. Follow-up work by Humphries and Kerschbaum [103] showed that Zhang et al.’s solution suffered from prohibitively poor utility and provided an improved algorithm which was evaluated on the k -medians clustering problem. While our work also applies an evolutionary-based approach to solve clustering, our algorithm uses significantly less privacy budget. The population of PE-means is a set of centroids from which a single solution of k points is chosen in each iteration using a DP histogram with sensitivity 1, following the previous work in PE [134, 135, 193, 201]. Humphries and Kerschbaum instead evolve a population where each candidate is a solution of k points and apply the exponential mechanism on the clustering loss (with sensitivity proportional to the data domain) to select multiple candidates in each iteration, using significantly more privacy budget.

4.6 Conclusion

We have shown that the PE algorithm is well-suited to the problem of k -means clustering. Our benchmark shows that PE-means and HDPE-means offer state-of-the-art performance on many synthetic and real clustering datasets. Furthermore, the design of PE-means is of independent interest, offering solutions to vote-splitting and signal-to-noise ratio management that can be extended to other modalities of PE. In future work, we will study variation APIs that more efficiently search high-dimensional Euclidean spaces.

Table 4.1: A summary of results over all datasets (real datasets highlighted in bold) showing the AUC of the clustering loss across ϵ values with the best approach highlighted for each dataset. The final column shows the relative improvements of PE-based techniques over the state-of-the art baseline for that dataset, with an average improvement of 26% over all datasets for PE-based approaches.

Dataset	N	d	k	Non-Priv	PE-means	HDPE-means	Google-LSH	FastLloyd	DP-Lib	Icml17	PE Improv.
birch2	25000	2	100	0.0001	0.0003	-	0.0658	0.0031	0.0124	0.0161	+90.92%
iris	150	4	3	0.1361	0.2894	-	0.6892	0.3979	1.3048	5.5766	+27.28%
adult	48842	6	3	0.0056	0.0056	-	0.0076	0.0155	0.0213	0.0113	+25.55%
mnist	70000	84	10	0.1958	0.2920	0.2169	0.2173	0.5822	0.5315	0.7904	+0.16%
letter	20000	16	26	0.2481	0.2852	-	0.3187	0.3963	0.4341	0.6277	+10.50%
gas	36733	12	6	0.1026	0.1081	-	0.1121	0.1561	0.1761	0.1853	+3.60%
g2_4	2048	4	2	0.4650	0.4761	-	0.5465	0.4666	0.5468	0.8802	-2.05%
g2_16	2048	16	2	0.7677	0.7800	-	0.8391	0.8687	1.1816	4.7489	+7.04%
g2_64	2048	64	2	1.0956	1.1538	1.1295	1.2478	1.9647	2.6608	27.5361	+9.49%
g2_128	2048	128	2	1.3534	1.7237	1.4055	1.5906	2.6425	4.4632	74.0978	+11.64%
scale_4_4	10000	4	4	0.0821	0.0814	-	0.1190	0.1272	0.1381	0.1772	+31.54%
scale_4_16	10000	16	4	0.3013	0.3147	-	0.3392	0.3721	0.4114	0.5948	+7.24%
scale_4_64	10000	64	4	0.4966	0.5326	0.5340	0.5430	0.5863	0.6622	2.5354	+1.92%
scale_4_128	9999	128	4	0.5731	0.6179	0.5940	0.5987	0.6282	0.7481	8.5885	+0.79%
scale_16_4	10000	4	16	0.0302	0.0335	-	0.0555	0.0864	0.0976	0.1525	+39.63%
scale_16_16	10000	16	16	0.2110	0.2452	-	0.2880	0.3624	0.3778	0.5679	+14.87%
scale_16_64	10000	64	16	0.4824	0.5884	0.5757	0.5823	0.6371	0.6949	2.5353	+1.14%
scale_16_128	10001	128	16	0.5410	0.6131	0.6023	0.5992	0.6294	0.7612	8.6815	-0.50%
scale_64_4	10002	4	64	0.0029	0.0051	-	0.0214	0.0306	0.0381	0.0847	+76.08%
scale_64_16	10001	16	64	0.1126	0.1607	-	0.1902	0.2393	0.2771	0.5238	+15.54%
scale_64_64	10001	64	64	0.4487	0.5518	0.5567	0.5453	0.5827	0.6913	2.5620	-1.20%
scale_64_128	10016	128	64	0.5206	0.6874	0.6232	0.6299	0.6421	1.0248	8.6433	+1.07%
sklearn_4_4	20000	4	4	0.0777	0.0778	-	0.1305	0.1777	0.2355	0.1240	+37.22%
sklearn_16_4	20000	4	16	0.0459	0.0445	-	0.0979	0.0990	0.1409	0.2601	+54.59%
sklearn_64_4	20000	4	64	0.0343	0.0379	-	0.1289	0.0818	0.1191	0.3227	+53.68%
sklearn_4_16	20000	16	4	0.0950	0.0955	-	0.1395	0.7110	0.5049	0.6074	+31.50%
sklearn_16_16	20000	16	16	0.0606	0.0668	-	0.1596	0.7160	0.5809	1.4670	+58.16%
sklearn_64_16	20000	16	64	0.0642	0.1183	-	0.3872	0.7631	0.8720	1.7704	+69.43%
sklearn_4_64	20000	64	4	0.0954	0.0993	0.0974	0.1090	1.4623	0.8614	2.7731	+10.60%
sklearn_16_64	20000	64	16	0.0847	0.1192	0.1189	0.2258	1.7475	1.3716	3.1765	+47.34%
sklearn_64_64	20000	64	64	0.0823	0.3070	0.4678	0.7878	2.2002	2.2456	3.1698	+61.03%
sklearn_4_128	20000	128	4	0.1054	0.1333	0.1097	0.1186	1.7604	1.1632	5.6663	+7.49%
sklearn_16_128	20000	128	16	0.0919	0.1942	0.1562	0.2838	2.3038	1.9742	4.9881	+44.97%
sklearn_64_128	20000	128	64	0.0933	0.5740	0.7190	0.9675	2.7204	2.9608	4.9597	+40.67%

Chapter 5

Interpreting the Error of Differentially Private Median Queries through Randomization Intervals

This chapter is adapted from an extended abstract available on arXiv and presented at the TPDW workshop [104]. For this case study, we consider a party with direct access to a sensitive dataset who wishes to publish private statistics about that dataset using differential privacy. In this specific system, we assume the audience of the statistics has a very limited understanding of differential privacy. Thus, without modification, existing differentially private algorithms have a usability bottleneck: the audience cannot interpret the effects of privacy on those statistics. For the private alternative to be deployed here, meaningful ways of communicating the effect of privacy on the statistic without degrading overall utility are needed. We propose one such way for the case of median queries in the following chapter.

5.1 Introduction

A core functionality of database management software is allowing analysts to query the database to learn aggregate statistics. Naively allowing analysts to obtain unperturbed statistics can open up a lucrative attack surface over sensitive data [60]. A popular defence to such attacks is Differential Privacy (DP) [69]. By adding a calibrated noise to each query result, the data owner enjoys a formal guarantee of privacy over the result of the queries. A

challenge with DP is that it requires a certain level of expertise to apply correctly, as well as to interpret the results of a DP query.

In this work, we focus on the problem of interpreting the level of error introduced by DP to satisfy privacy. One method to help analysts understand this error is to return a confidence interval alongside the noisy result. Specifically, an upper and lower bound on the noisy statistic such that the true answer is contained in the interval with high confidence (e.g., 95%). To avoid confusion with confidence intervals from statistics that bound sampling error, we introduce the term *randomization intervals (RI)* for a confidence interval that bound only the randomness of the DP mechanism itself. These randomization intervals allow the analyst to reason about the error of the query without understanding the DP mechanism. For example, if the interval is small, the analyst can be confident that the result is accurate. Conversely, consider an analyst conducting a count query to determine the number of people in the dataset with a specific attribute. If the RI for this query contains zero, the user may not trust that there are any people in the dataset with this attribute.

For queries such as sums and counts, where the randomness is independent of the input, one can use a tail bound to compute the RI at no additional privacy cost. However, this method does not extend to more general queries, such as the median, that require a data-dependent DP noise distribution (e.g. the exponential mechanism) for good utility. Sun et al. were the first to study the problem of randomization intervals for the median (and other queries) [187]. Instead of returning a high utility estimation for the median along with an estimated RI, Sun et al. focus on estimating the RI first, and then post-process the median as the average of the RI bounds. The challenge with this approach is that the utility of the median becomes significantly degraded as estimating an RI has an inherently larger error factor to ensure high confidence. Furthermore, the private data is not guaranteed to be evenly distributed within the RI, making the center of the RI a poor estimation of the median.

We introduce **PostRI**, a new approach to computing the RI alongside the median, while maintaining the high utility of the classic DP median computation. **PostRI** first computes the DP median using an unmodified exponential mechanism approach. Then, using our single-shot, low-sensitivity utility function, we estimate the RI using a tunable amount of privacy budget left over from the median. **PostRI** outputs an RI of similar width to Sun et al. [187], while significantly improving the error of the median query itself. We prove the privacy and correctness of **PostRI** and derive optimal values for hyperparameters such as the ratio of privacy budget between the median and RI. We evaluate **PostRI** over real-world datasets and find a 14%-850% improvement in the average median error over [187], while maintaining approximately the same or a slightly larger RI width, depending on the

dataset.

5.2 Background

Differential privacy (DP) [69] guarantees that an algorithm's output is approximately the same, regardless of the participation of any single user. More formally, differential privacy can be defined as follows.

Definition 5.2.1 (Differential Privacy (DP)). *A randomized algorithm $M : \mathcal{D} \rightarrow \mathcal{O}$ is ϵ -DP if for any pair of neighboring databases $D, D' \in \mathcal{D}$, and all $O \subseteq \mathcal{O}$ we have*

$$\Pr[M(D) \in O] \leq e^\epsilon \Pr[M(D') \in O].$$

We use the bounded neighbouring definition where datasets are neighbours if they differ in the replacement of a single record, $|D \cap D'| = n - 1$. Note that if we apply a DP mechanism(s) sequentially, the privacy parameters are composed through summation or more advanced methods [70].

The exponential mechanism, introduced by McSherry and Talwar, is a general-purpose DP mechanism that maximizes a given utility function privately [149].

Definition 5.2.2 (Exponential Mechanism [149]). *Given privacy budget ϵ and utility function $u : \mathcal{D} \times \mathcal{Y} \rightarrow \mathbb{R}$ with sensitivity $\Delta_u := \max_{D, D' \in \mathcal{D}, y \in \mathcal{Y}} |u(D, y) - u(D', y)|$, the exponential mechanism $\text{Exp}_u(D, \epsilon)$ outputs a sample $y \in \mathcal{Y}$, with the following probability*

$$\Pr[y] = \frac{\exp\left(\frac{\epsilon u(D, y)}{2\Delta_u}\right)}{\sum_{i \in \mathcal{Y}} \exp\left(\frac{\epsilon u(D, i)}{2\Delta_u}\right)}. \quad (5.1)$$

The exponential mechanism, as defined above, guarantees ϵ -differential privacy [149, Theorem 6].

5.2.1 Problem Setup

Definition 5.2.3 (Randomization Interval). *Given a dataset D of size n , privacy budget ϵ , and a failure probability β , we wish to output a triple (l, o, u) called a Randomization Interval (RI) such that:*

- *outputting (l, o, u) satisfies ϵ -DP.*
- *o is a DP estimate of the true median m of dataset D .*
- *With probability $1 - \beta$, $m \in [l, u]$.*

Intuitively, we wish to give a lower and upper bound on the noisy median estimate to make the error of the noisy median more interpretable. Another way to state this goal is that we are computing a confidence interval with respect to the error introduced by differential privacy (we clarify the relation to related work on DP confidence intervals in Section 5.6). We consider computing the true median to be deterministic in this work.

For simplicity, we assume $\mathcal{Y}$ is an integer domain of size N , denoted $\mathcal{Y} = [N] = \{0, 1, \dots, N\}$. We also assume the utility function u is 1-Lipshitz, that is, $|u(D, y) - u(D, y + 1)| \leq \ell = 1$ for any $D \in \mathcal{Y}^n, y \in \mathcal{Y}$. In the case of the median, this implies that the dataset D contains no repeated elements. For datasets that do not satisfy this assumption, we follow Sun et al. [187] and remap the data to an expanded domain of size $N \cdot n$. The de-duplicated domain can be found by mapping $D \in [N]^n$ to $\tilde{D} \in [n \cdot N]^n$, where the k repetitions of an element $x \in D$ are mapped to consecutive elements of the new domain: $nx, nx + 1, \dots, nx + k - 1$.

5.3 PostRI

In this section, we first overview the design of PostRI in the two main steps, the median estimation and then the randomization interval. We then analyze the privacy, correctness, and utility of the complete algorithm and finally discuss the optimal hyperparameters.

5.3.1 Median Mechanism

The first step in our algorithm is to derive a differentially private median using a standard approach [130]. We give the details of this algorithm for completeness. Since PostRI operates in two parts, we must split the privacy budget between this median computation and the randomization interval. We denote the privacy budget of the median as ϵ_1 and the privacy budget of the RI as ϵ_2 , where $\epsilon = \epsilon_1 + \epsilon_2$ and discuss how to set these parameters in Section 5.4.1.

The utility function we will use for the median is

$$u(D, y) = -|R(D, y) - \frac{|D|}{2}| \quad (5.2)$$

where $R(D, y)$ computes the number of data points less than or equal to y after sorting the dataset D .

The utility function $u(D, y)$, has sensitivity of 1, and thus applying the exponential mechanism satisfies ϵ_1 -DP [149, Theorem 6]. In the case where multiple values in the domain $\mathcal{Y}$ have the same utility, it is common to weight the probability of selection by the number of elements with this utility, and then randomly select a candidate after [89]. We do this for the median, since there can be many domain values between each data point, all with the same rank.

Median Utility

There is a well-established utility bound on the exponential mechanism that we can apply to get a utility bound on the median [70].

Theorem 5.3.1. Let $y^*(D) = \operatorname{argmax}_{y \in \mathcal{Y}}(u(D, y))$. Then, with probability $1 - \beta_1$ the following statement holds [70]:

$$u(D, o) \geq u(D, y^*(D)) - \gamma_1 \quad (5.3)$$

where

$$\gamma_1 = \frac{2\Delta_u}{\epsilon_1} \log \frac{N}{\beta_1} \quad (5.4)$$

5.3.2 RI Mechanism Design

Since there exists a bound on the error of the exponential mechanism (Theorem 5.3.1), constructing a randomization interval for the median seems simple. However, this error bound is in terms of the utility function, which presents several problems. First, we do not have the utility value of the outputted median, as publishing the utility requires additional privacy budget. Second, even if we publish the utility, the value is relative and is not useful without knowledge of the private dataset. In the case of the median, a randomization interval using the utility bound would only tell us the rank error. For example, we could bound the median in a range of up to ten dataset positions, but depending on the dataset,

the closest ten data points could be numerically very far away from the median. Thus, we want to give a randomization interval in the data domain.

A strawman solution could be to compute the utility bound and then map it back onto the data domain. However, this would use a significant amount of privacy budget, paying both to compute the utility bound and select the values from the data domain. We instead develop a novel utility function to output the randomization interval in a single application of the exponential mechanism.

In order to save additional privacy budget, we first reformulate the randomization interval problem as the private selection of a single value b (rather than a separate upper and lower bound).

Simplified Randomization Interval Problem: Given an output median o , find a minimal b such that with probability $1 - \beta_2$, while satisfying ϵ_2 -DP.

$$m \in [o - b, o + b]$$

where m is the ground truth median.

Using this simplified formulation, we can then design a utility function for the optimal b . Our first step is to create a helper function f that measures the worst-case coverage of a given b in terms of rank using dataset D . We define this function as

$$f_b(D) = \min \left(|R(D, o + b) - R(D, o)|, |R(D, o) - R(D, o - b)| \right).$$

Intuitively, f computes the smallest rank distance from the interval boundary to the median. We note that we chose the above function for two reasons. First, it allows us to consider both sides of the interval in one shot. Second, it has sensitivity one (shown in Lemma 5.3.2), making it very efficient to compute privately.

Recall, the goal is to construct a valid randomization interval by making b large enough that the true median m is contained in the interval with high probability. To determine this width, we can use the utility bound in Theorem 5.3.1. Since we designed the helper function to measure rank distance, and the utility function of the median measures rank distance, we can simply find the minimum b such that $f_b(D) > \gamma_1$. A natural way to privately select such a b would be to use the sparse vector technique (SVT) [70]. SVT sequentially evaluates a given set of queries and halts (and outputs the query index) when the first query value exceeds this threshold. In our case, the set of queries would be defined by evaluating the

helper function f over a set of b values and the threshold would be γ_1 . However, it was shown by Lyu et al. [143] that in a non-interactive setting such as this (the queries are known ahead of time), replacing SVT with the exponential mechanism gives better utility¹. Thus, in our work, we apply the exponential mechanism using a utility function that is the absolute difference between the query set and the threshold.

We note that in addition to the threshold γ_1 , we must account for two additional sources of error. The first is due to the fact that we are selecting b using another exponential mechanism. We must ensure correctness under the worst-case error of this second exponential mechanism in selecting the error of the first. Thus, we must incorporate a second error bound similar to Theorem 5.3.1 but with the following constant.

$$\gamma_2 = \frac{2\Delta_q}{\epsilon_2} \log \frac{N}{s\beta_2} \quad (5.5)$$

where s is a quantization parameter determining the domain set we choose for b . This quantization is the second source of error we must account for. Specifically, if we consider all possible b values, we have a larger domain that negatively affects error; if we pick a smaller domain, we must account for the quantization error and the new domain size. We define the domain set to be $b \in \{s, 2s, \dots, \lfloor \frac{s \cdot N}{s} \rfloor\}$. This gives a domain size of N/s and introduces a quantization error. Namely, b can be at most s away from an optimal value. Since b is in the data domain and the utility is in the rank domain, we multiply s by ℓ (the Lipschitz coefficient assumed to be 1 if no repeated data elements) to obtain a bound on this error in terms of utility.

Putting all of this together, we get our final utility function.

$$q(D, b) = -|f_b(D) - \gamma_1 - \gamma_2 - s \cdot \ell| \quad (5.6)$$

We apply the exponential mechanism on q to obtain an ϵ_2 -DP RI mechanism.

5.3.3 Privacy Analysis

Before proving the privacy of PostRI, we first prove the following lemma.

Lemma 5.3.2. *f has sensitivity 1. Formally:*

$$\max_{\substack{D, D' \in \mathcal{D} \\ b \in \{s, 2s, \dots, \lfloor \frac{s \cdot N}{s} \rfloor\}}} |f_b(D) - f_b(D')| \leq 1 \quad (5.7)$$

¹Lyu et al. consider top- k queries, but we find the same result holds for our threshold queries

Proof. Recall $f_b(D) = \min(|R(D, o+b) - R(D, o)|, |R(D, o) - R(D, o-b)|)$. w.l.o.g assume that $R(D, o-b) \leq R(D, o) \leq R(D, o+b)$. Then, the term $|R(D, o-b) - R(D, o)|$ is equivalent to the number of data points that fall in the interval $(o-b, o]$. Replacing a data point can, in the worst case, move a point in or out of this interval, implying a sensitivity of one. A symmetric argument holds for $|R(D, o+b) - R(D, o)|$. Finally, we consider the min. The min either remains unchanged or the min swaps under the replacement of a record. In either case, the output can change by at most one, as both terms can change by at most one. $\square$

We can now state and prove the privacy of our mechanism.

Theorem 5.3.3. q has a sensitivity of 1 and thus applying the exponential mechanism on it satisfies ϵ_2 -DP.

Proof. We observe that the sensitivity of q is equivalent to the sensitivity of f since all other terms are constant over neighbouring datasets. Then, by Lemma 5.3.2 and the privacy properties of the exponential mechanism, the result follows. $\square$

5.3.4 Correctness

In this section, we prove the correctness of **PostRI**. Namely:

Theorem 5.3.4. For a given private median estimation o and randomization interval $[o - \hat{b}, o + \hat{b}]$ output using **PostRI**, with probability $1 - \beta_1 - \beta_2$, we have $R(D, o - \hat{b}) \leq n/2 \leq R(D, o + \hat{b})$

Proof. With probability $1 - \beta_1$ since o is the output of the exponential mechanism, we have:

$$u(D, o) \geq u(D, y^*(D)) - \gamma_1 \quad (5.8)$$

subbing in the definition of the median utility function (5.2), gives the following

$$|R(D, o) - n/2| \leq \gamma_1 \quad (5.9)$$

which implies

$$R(D, o) - \gamma_1 \leq n/2 \leq R(D, o) + \gamma_1. \quad (5.10)$$

For a given output $\hat{b}$ of the second exponential mechanism, with probability $1 - \beta_2$, we have the following statement

$$q(D, \hat{b}) \geq q(D, b^*(D)) - \gamma_2 - s \cdot \ell \quad (5.11)$$

which gives

$$|f_{\hat{b}}(D) - \gamma_1 - \gamma_2 - s \cdot \ell| \leq \gamma_2 + s \cdot \ell. \quad (5.12)$$

W.l.o.g assume that $R(D, o - \hat{b}) \leq R(D, o) \leq R(D, o + \hat{b})$. We consider two cases $R(D, o) \geq n/2$ and $R(D, o) < n/2$. In the first case, if $R(D, o) \geq n/2$ then by assumption we have $R(D, o + \hat{b}) \geq R(D, o) \geq n/2$. We must show $R(D, o - \hat{b}) < n/2$. Expanding Eqn 5.12 we get the following

$$-\gamma_2 - s \cdot \ell \leq f_{\hat{b}}(D) - \gamma_1 - \gamma_2 - s \cdot \ell \quad (5.13)$$

where the absolute value is omitted following our initial assumption. Then we get

$$-\gamma_2 - s \cdot \ell \leq R(D, o) - R(D, o - \hat{b}) - \gamma_1 - \gamma_2 - s \cdot \ell. \quad (5.14)$$

We note that this holds regardless of which term is the min in $f_{\hat{b}}(D)$. Then rearranging and subbing in Eqn 5.10, we get

$$R(D, o - \hat{b}) \leq R(D, o) - \gamma_1 \leq n/2. \quad (5.15)$$

In the second case, if $R(D, o) < n/2$ then by assumption we have $R(D, o - \hat{b}) \leq R(D, o) < n/2$. We must show $R(D, o + \hat{b}) > n/2$. Recall that expanding Eqn 5.12 we get the following

$$-\gamma_2 - s \cdot \ell \leq f_{\hat{b}}(D) - \gamma_1 - \gamma_2 - s \cdot \ell. \quad (5.16)$$

Then we get

$$-\gamma_2 - s \cdot \ell \leq R(D, o + \hat{b}) - R(D, o) - \gamma_1 - \gamma_2 - s \cdot \ell. \quad (5.17)$$

We similarly note that this holds regardless of which term is the min in $f_{\hat{b}}(D)$. Finally, rearranging and subbing in Eqn 5.10, we get

$$R(D, o + \hat{b}) \geq R(D, o) + \gamma_1 \geq n/2. \quad (5.18)$$

Taking the union bound over the probability of Eqn 5.8 and Eqn 5.11, the result follows. $\square$

5.4 Utility Analysis

In this section, we give a some high-level utility claims for **PostRI**. We note that a formal utility bound can not be derived for **PostRI**. This is because our utility function takes a minimum over two terms in order to give a lower bound on the width of the RI for

correctness. In order to prove an upper bound on the width, we would need to consider both terms in the minimum which increases sensitivity.

To compare the utility of our work to that of Sun et al. [187], we first state the utility bound from their paper. Namely, for any $y \in [L(D), R(D)]$ (in the RI), they show that

$$u(D, y) \geq u(D, y^*(D)) - \frac{17\Delta_u}{\epsilon} \log \frac{2N}{\beta} - 2\ell \quad (5.19)$$

To roughly compare the utilities of the work, we first define a constant term of factors common to both approaches.

$$\eta = \frac{2\Delta_u}{\epsilon} \log \frac{2N}{\beta} \quad (5.20)$$

Using η , we make the following observations.

Claim 5.4.1. *Assuming the best case where the exponential mechanisms return the candidates with optimal utility, the width of the RI for both approaches will be approximately 8η .*

Proof Sketch. The utility function of Sun et al. is $u(D, y) - u(D, y^*(D)) - s - \ell$, where $u(D, y^*(D)) = 0$, $s \approx 4\eta$, and we will ignore $\ell = 1$. This means the distance from the median to the lower bound is approximately 4η . The upper bound is similar, resulting in a total width of 8η .

Our utility function is $|f_b(D) - \gamma_1 - \gamma_2 - s \cdot \ell|$ where $\gamma_1 \approx 2\eta$, $\gamma_2 \approx 2\eta$, and we ignore the ℓ term. This gives an optimal helper function f of 4η (the worst-case distance to either the upper or lower bound), which implies a total width of 8η . $\square$

Claim 5.4.2. *Assume the worst case where the exponential mechanisms return the candidates with the worst possible utility defined by Theorem 5.3.1. Then Sun et al. have an RI width of 16η and PostRI has an RI width of 12η .*

Proof Sketch. Building on Claim 5.4.1, we can assume the error of each application of the exponential mechanism in Sun et al. is 4η by applying the utility bound of the exponential mechanism with an extra factor of 2 from splitting epsilon over the lower and upper bound and another 2 from Lemma 3.3 of Sun et al. [187]. This means the lower bound estimates the distance of 4η from the median with at worst a 4η error, resulting in a total distance of 8η . Similarly, for the upper bound, giving the total width of 16η .

Our work builds the RI using the estimated median as the center. Then we estimate a width of 4η (as discussed in Claim 5.4.1) with error at most $\gamma_2 \approx 2\eta$. This leads to a total distance of 6η per side and an overall distance of 12η . $\square$

Claim 5.4.3. *Assume the worst case where the exponential mechanisms return the candidates with the worst possible utility defined by Theorem 5.3.1. Then Sun et al. have a median error of 4η and *PostRI* has a median error of 2η .*

Proof Sketch. Building on Claim 5.4.2, if we assume the lower bound is the worst possible distance of 8η and in the worst case, the upper bound is the smallest possible distance of 0 from the true median (estimating 4η distance with error 4η that cancel each other out). Then the average of the lower and upper bounds is 4η away from the true median.

Our work simply estimates the median with error at most $\gamma_1 \approx 2\eta$. $\square$

5.4.1 Hyperparameter Selection

PostRI has two hyperparameters that can be varied. The first is the split between ϵ_1 and ϵ_2 . That is, more budget can be allotted to reduce the error of the median or to shorten the length of the RI. However, we note that one can not arbitrarily shorten the RI as it inherently depends on the error of the median in our approach. We derive the optimal split between ϵ_1 and ϵ_2 to give the shortest possible RI width by minimizing the following error.

$$\gamma_1 + \gamma_2 + s \cdot \ell \quad (5.21)$$

using Lagrange multipliers with the condition that $\epsilon_1 + \epsilon_2 = \epsilon$. Let

$$L = \frac{2}{\epsilon_1} \log \frac{N}{\beta_1} + \frac{2}{\epsilon_2} \log \frac{N}{s\beta_2} + s \cdot \ell + \lambda(\epsilon_1 + \epsilon_2 - \epsilon). \quad (5.22)$$

Then, differentiating, we get

$$\frac{\partial L}{\partial \epsilon_1} = -\frac{2}{\epsilon_1^2} \log \frac{N}{\beta_1} + \lambda \quad (5.23)$$

$$\frac{\partial L}{\partial \epsilon_2} = -\frac{2}{\epsilon_2^2} \log \frac{N}{s\beta_2} + \lambda. \quad (5.24)$$

Setting these equal and rearranging gives us the following:

$$\epsilon_1^2 = \epsilon_2^2 \cdot \frac{\log \frac{N}{\beta_1}}{\log \frac{N}{s\beta_2}} \quad (5.25)$$

Taking the square root gives us the final result:

$$\epsilon_1 = \epsilon_2 \sqrt{\frac{\log \frac{N}{\beta_1}}{\log \frac{N}{s\beta_2}}} \quad (5.26)$$

Table 5.1: Comparison of Median RI on Real Datasets with $\epsilon = 1$ and $\beta = 0.01$.

Dataset	Technique	Median Error	Average RI Width	Correctness
Bank	Ours	0.06 ($\pm$ 0.24)	14.19 ($\pm$ 0.67)	1.00
	[187]	0.57 ($\pm$ 0.62)	13.63 ($\pm$ 0.55)	1.00
Adult	Ours	32.40 ($\pm$ 28.61)	1264.00 ($\pm$ 74.33)	1.00
	[187]	166.88 ($\pm$ 17.23)	1146.56 ($\pm$ 17.11)	1.00
Airplane	Ours	7.88 ($\pm$ 4.81)	13.13 ($\pm$ 2.58)	1.00
	[187]	9.00 ($\pm$ 0.00)	9.00 ($\pm$ 0.00)	1.00

Optimal Step Size The second parameter is the step size s , which determines the domain of b . We derive the optimal parameter setting for s such that the RI length is minimized. Let us assume that the RI width of our algorithm is $\gamma_1 + \gamma_2 + s \cdot \ell$. We assume all other variables are constant. Finally, we assume we always conduct queries for $b \in \{s, 2s, \dots, \lfloor \frac{sN}{s} \rfloor\}$. Then, ignoring constants (the median error), the width of the RI is

$$\gamma_2 + s \cdot \ell = \frac{2\Delta_q}{\epsilon_2} \log \frac{N}{s\beta_2} + s \cdot \ell \quad (5.27)$$

The first derivative of this w.r.t. s is

$$-\frac{2\Delta_q}{\epsilon_2 s} + \ell \quad (5.28)$$

Solving for s gives the optimal way to set this parameter

$$s = \frac{2\Delta_q}{\epsilon_2 \ell} \quad (5.29)$$

where ℓ is the Lipschitz bound on the RI utility function.

We note a circular dependence between the optimal choice of ϵ_1 , ϵ_2 , and s . In practice, we find that substituting these equations into each other iteratively converges to a stable value after a few iterations.

5.5 Preliminary Experimental Results

To evaluate our method PostRI, we conduct a study over three real-world datasets and compare with the existing approach by Sun et al. [187].

Implementation Details Our implementation is based on the code of Sun et al. [187]: https://github.com/PrivateCI/DP_CI. We note that running their code unmodified did not reproduce the results Sun et al.’s paper exactly. We modified the domain size in their code to $N = 10^8$ to be consistent across both approaches. We also note that the parameter s in Sun et al.’s work is set to $\frac{9\Delta_u}{\epsilon} \log \frac{2N}{\beta}$ in the paper, but $\frac{8\Delta_u}{\epsilon} \log \frac{2N}{\beta}$ in the code. We follow their code as it gives a lower error and matches the settings of our work. We also fixed a couple of off by one errors in Sun et al.’s code. Our code can be found at <https://github.com/Timliuw/Randomization-Intervals>.

Experimental Setup We focus on the “balance” attribute (with values in range $[-8019, 102127]$ and true median value of 448) for the Banking dataset, the “fnlwgt” attribute (with values in range $[12285, 1490400]$ and true median value of 178144.5) for the Adult dataset, and the “capacity” attribute (with values in range $[4, 396]$ and true median value of 162) for the Airplane dataset in the evaluation. We fix s following Eqn 5.29. We use the average median error (i.e., numerical distance from true median), the average RI width (i.e., the distance between the lower and upper bounds), and the observed correctness (i.e., the percentage of how many times the true median is inside the reported RI) over 100 runs as evaluation metrics.

Results Table 5.1 shows the comparison results with privacy budget $\epsilon = 1$ and $\beta = 0.01$. Our PostRI method (default setting, with $\epsilon_1 = \epsilon_2 = \frac{1}{2}\epsilon$) improves the average median error over [187] by 14%-850% with a moderately wider RI width by 3%-35%. The RI results reported by both methods contain the true median 100% of the time in our experiments, implying an observed correctness rate of 1.

We also measure the average median error and RI width with varying privacy budgets ($\epsilon = 0.25, 0.5, 1, 2, 4$) across datasets. We compare [187] and PostRI with different budget splitting: default ($\epsilon_1 = \epsilon_2$), optimal (Eq. 5.26), and median-focused ($\epsilon_1 = 9\epsilon_2$). As shown in Fig. 5.1-5.3, our PostRI has lower median error over all privacy regimes on these datasets and yields a comparable RI width as the method in [187], which matches our theoretical analysis (Section 5.4). The shaded areas depict the standard deviation across 100 runs. Notably, an advantage of our approach is being able to tune the privacy budget split in favour of the median. Experiments on the Banking dataset (Fig. 5.1) show that the median-focused split (i.e., 90% of the budget is spent on the median) can achieve the most accurate result on high privacy regimes, albeit with a wider RI width, compared to the default setting and the optimal split that minimizes the overall error.

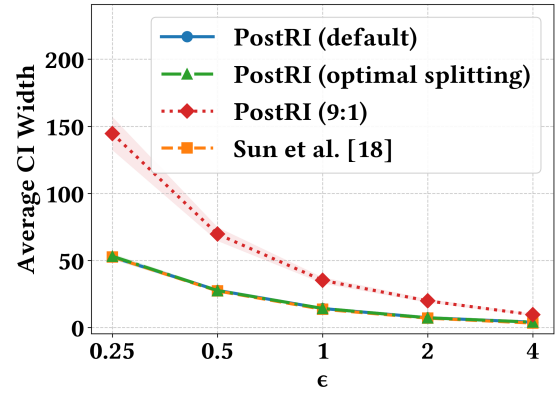
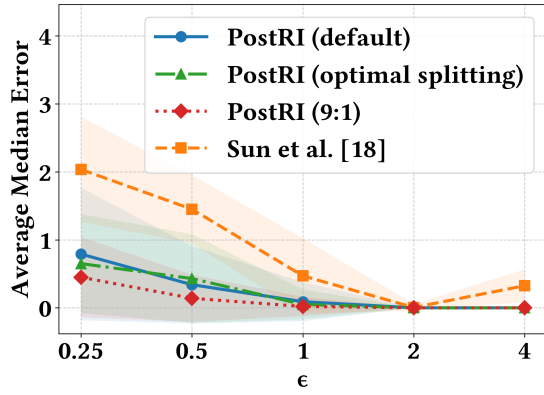


Figure 5.1: Average median error and RI length vs. varying privacy budget on the Banking dataset

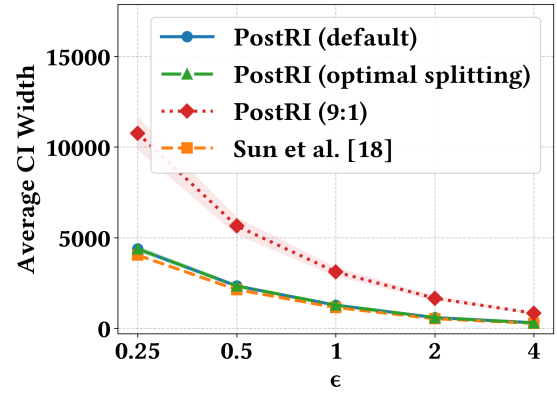
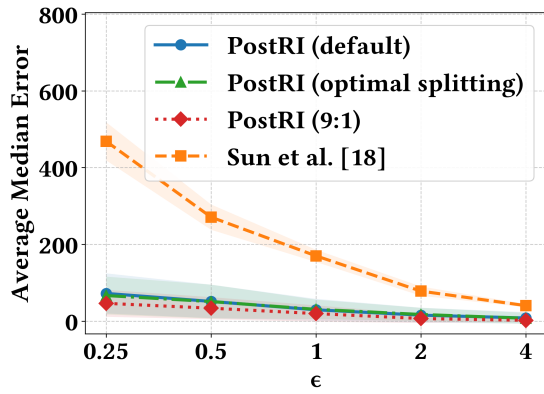


Figure 5.2: Average median error and RI length vs. varying privacy budget on the Adult dataset

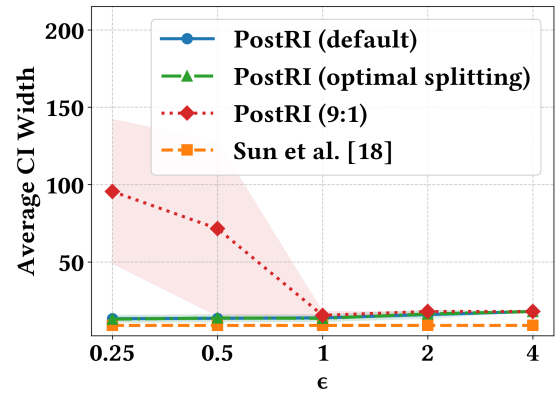
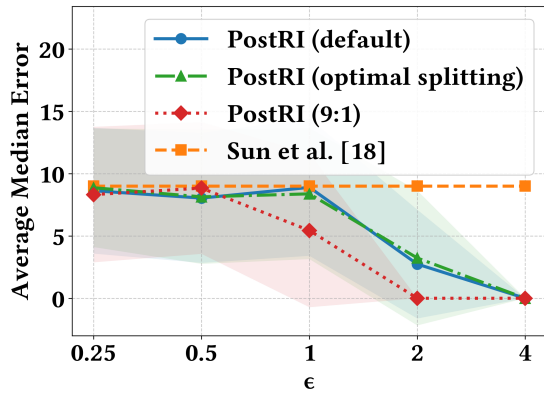


Figure 5.3: Average median error and RI length vs. varying privacy budget on the Airplane dataset

5.6 Related Work

There has been a significant amount of work done on expressing the privacy-utility trade-off present in any DP mechanism to a non-expert. An approach can be to provide a set of benchmark utility scores for common privacy parameter settings and datasets [96, 200]. A more usable approach is to provide scores as an interactive visualization tool [138, 158, 169, 200], so that users can explore possible privacy settings.

Our work looks at expressing the privacy utility trade-off using randomization intervals. We note that although randomization intervals provide a bound on the DP noise with high confidence, they do not provide a bound on the variance from the data (sample variance). Most prior work on DP confidence intervals focuses on bounding this error, without considering the error from the noise itself [38, 44, 47, 63, 68, 133].

In our work, we treat finding bounds for a confidence interval as a maximizing a utility function problem. This then leads us to using an exponential mechanism-based design. Sun et al. [187] also take this approach. They provide confidence intervals for the median and other statistics using both exponential mechanism and SVT based approaches. Their approach first estimates valid upper and lower bounds for the statistic, before taking the average of the values.

Drechsler et al. [62] consider solutions that estimate confidence intervals using both the exponential mechanism and post-processing an estimated DP CDF. Their solutions aim to account for both sources of randomness (sample error and privacy noise). However, in both of their approaches, they do not directly estimate the median itself.

5.7 Concluding Remarks

In this work, we improve the state-of-the-art in computing randomization intervals for private median queries. Our solution **PostRI** uses a novel utility function to enable a high utility median estimation along with a narrow randomization interval. In the future, we look to extend this idea to other statistics.

Chapter 6

Investigating Membership Inference Attacks under Data Dependencies

This chapter is adapted from work that previously appeared at the IEEE Computer Security Foundations Symposium (CSF) [106]. This case study in auditing highlights the need to verify that a theoretical privacy claim actually holds in the scenarios encountered in practice. For this particular system, we assume a central party collects data from a population of users with no privacy protection. The central party then trains a machine learning model over the user data, but uses differential privacy in the training process to offer privacy protection to the users. For this scenario, what privacy protections are claimed and how these are communicated are critical. In this chapter, we analyze how two seemingly similar notions of privacy (protection against membership inference and differential privacy) are related but different. Specifically, the central party can always claim differentially private guarantees. However, claiming protection against a membership inference attack is more nuanced and requires making additional assumptions about the data collection.

6.1 Introduction

Machine learning (ML) is increasingly used to make predictions on privacy-sensitive data. In recent years, large tech companies such as Google and Amazon have begun to offer machine learning as a service to the general public through their cloud platforms. Although these systems can yield new and interesting insights in a variety of fields, machine learning models trained on sensitive data also present a lucrative attack surface for adversaries. In

a Membership Inference Attack (MIA) [181, 205], an adversary attempts to identify the data that was used to train an ML model (i.e., its members). Successful MIAs violate the privacy of individuals and pose a significant threat to unprotected ML models [181].

Differentially private (DP) mechanisms can be applied to an ML model during the learning process [11, 26, 41, 181, 205, 206] to limit the effect that a single data point can have on the model’s output. Training ML models with DP mechanisms is becoming more common with popular libraries such as TensorFlow [2] and PyTorch [3] offering several DP training algorithms.

Previous work has provided experimental and theoretical evidence of DP training as a defence against MIAs [111, 131, 181, 205]. Yeom et al. [205] prove that DP training algorithms ensure a *theoretical upper bound* on the privacy leakage caused by MIAs. This bound, later improved by Erlingsson et al. [112], offers strong protection for high DP levels (corresponding to small values of the DP parameter ϵ), but quickly weakens for large ϵ . Despite this, empirical evaluations of MIAs against DP-trained models [111, 112] suggest that large values of ϵ , theoretically regarded as low-privacy settings, provide sufficient protection against MIAs in the wild. Motivated by these findings, Murakonda and Shokri [155] developed a tool for tuning ϵ based solely on the empirical performance of MIAs, thus increasing the model’s utility at the expense of theoretical privacy guarantees. In short, this body of literature suggests that applying DP mechanisms to ML models, even in severely weakened privacy regimes, provides sufficient protection against known MIAs.

These works on membership inference all follow a similar evaluation methodology where members and non-members are sampled independently from the same distribution. However, this underlying assumption is not always representative of real data [18, 116, 156, 157, 174, 177, 178]. Furthermore, prior work has shown that data dependencies can lead to an overestimation of the privacy protection provided by DP [119, 120, 131, 136, 194]. Motivated by this, we conduct the first study of MIA performance on ML models trained with DP-SGD [11] *under data dependencies*.

We begin by studying the methodology from previous work on MIAs and identify the assumptions that underpin their theoretical findings [111, 162, 205]. Since there are no publicly known instances of MIAs against ML models in the wild, researchers are limited to simulating attacks using theoretical games between the model owner and attacker called *membership experiments* [205]. We show that previous membership experiments assume independence between training samples, which might not represent many membership inference cases in practice. To solve this, we propose a new membership experiment that loosens the restrictive independence assumption. We demonstrate that previous DP-based bounds on the adversary’s success do not hold in this more general setting. We emphasize

that this does not mean that DP is broken or flawed. Rather, the guarantees of DP mechanisms no longer apply when defining membership inference with data dependencies. Intuitively, this is because DP mechanisms limit the effect that a *single* sample has on the output. However, when training samples have dependencies, related samples can leak additional information about this single sample.

Prior work has shown the importance of studying MIA performance both theoretically and empirically [111, 112]. Thus, we also study the effect of data dependencies on the *empirical performance* of MIAs under different instantiations of our membership experiment. Using six real-world datasets, we evaluate the performance of off-the-shelf attacks. These attacks have black-box query access to the target model and are given no additional background information about the data dependencies we consider. Using these attacks, we investigate three different types of data dependencies. First, to study the extent to which data dependencies can affect MIAs, we artificially split a dataset into members and non-members using a clustering algorithm. Our results show that data dependencies can lead to perfect membership inference even in the presence of DP training. Second, we investigate the effects of an explicit bias in an attribute of the training set. In particular, we create a gender and an education bias in the training set, and see that MIA performance typically increases as the training set bias becomes more pronounced. Finally, we consider MIA performance on various real-world examples of data dependencies that occur during data collection. For example, we study the effect of all members being from a specific health region (or hospital) and non-members from all other regions. We also study the performance of MIAs when members and non-members come from the same source (the US Census) but are curated by different researchers.

In all of our evaluations with statistical dependencies among training samples we find a significant increase in MIA performance over the related work where all samples are Independent and Identically Distributed (IID) [110–112, 162, 181, 205]. This shows that the assumption of data independence among training samples underestimates the attack performance. Our empirical evaluation yields attack performance greater than the bounds of DP, which confirms that the currently known DP bounds do not apply once we consider dependent data, and thus DP mechanisms are not a cure-all solution for membership inference. This highlights the importance of considering data dependencies in future MIA evaluations and urges new research to develop better defences, and more realistic theoretical bounds on MIA performance.

Table 6.1: Notation

Notation	Description
$z = (x, y)$	Data point with feature vector x and label y
$\mathcal{Z}$	Space of all possible data points
S	Training set of size n ; $S \in \mathcal{Z}^n$
$\mathcal{D}$	Distribution of a data point (over $\mathcal{Z}$)
$\mathcal{D}$	Joint distribution of n data points (over $\mathcal{Z}^n$)
$[K]$	Set of integers from 1 to K
$\mathcal{A}$	Space of all possible machine learning models
A	Learning algorithm $A : \mathcal{Z}^n \rightarrow \mathcal{A}$
a	Instance of a trained model ($a \in \mathcal{A}$)
Att	Membership inference attack, outputs a bit
Adv	Membership advantage, $\text{Adv} \in [0, 1]$

6.2 Preliminaries

In this section, we summarize the concepts related to membership inference attacks and differentially private machine learning that are most relevant to our work. For reference, our notation is summarized in Table 6.1.

We use $z = (x, y)$ to denote an element or data sample, where x is its feature vector and y is its class or label. Let $\mathcal{Z}$ be the element space, i.e., $z \in \mathcal{Z}$. We use $S \in \mathcal{Z}^n$ to denote a training set that contains n elements $z \in \mathcal{Z}$. Let $\mathcal{D}$ be a probability distribution over $\mathcal{Z}$; $z \sim \mathcal{D}$ means that z is randomly sampled from $\mathcal{D}$, and $S \sim \mathcal{D}^n$ means that S consists of n independent samples from $\mathcal{D}$. We use $\mathcal{D}$ to denote a probability distribution over $\mathcal{Z}^n$, i.e., a *joint* distribution for all samples in a dataset; $S \sim \mathcal{D}$ means the dataset S is sampled from $\mathcal{D}$. We use $[K]$ to denote the set of integers from 1 to K . A training algorithm A is a (possibly randomized) function that takes a training set $S \in \mathcal{Z}^n$ and outputs a trained model $a \in \mathcal{A}$, where $\mathcal{A}$ is the space of trained models. We use $a = A(S)$ to denote that a is the model that results from applying the training algorithm A to the training set S . The goal of the trained model a is to solve a classification task; i.e., assign a label y to a feature vector x . In this work, we focus on neural networks trained with DP-SGD [11], which are a popular model choice for solving classification problems in machine learning. However, our

theoretical findings are generic and apply to other models as well.

6.2.1 Membership Inference Attacks

Though useful for solving classification tasks, machine learning models are subject to various privacy attacks. In this work we focus on the Membership Inference Attack (MIA), whose goal is to determine whether or not a specific data point z was included in the training set of a target model a . This attack is particularly dangerous when the model is trained on sensitive data, where an individual’s inclusion or exclusion in the dataset could reveal sensitive or compromising information to an attacker. MIAs are related to property inference attacks [19, 82, 145], but are fundamentally different. In a Property Inference Attack (PIA), the goal of the attacker is to infer some property of the training set that the model producer did not intend to share (i.e., a property common to all training set samples), whereas the goal of an MIA is to infer whether or not a particular sample was in the training set. We discuss the relationship to PIAs further in Section 6.5.

We use **Att** to denote a membership inference attack. Typically, an MIA receives the target model a and a data point z , and knows the training mechanism A and some statistical information about the training data (e.g., $\mathcal{D}$ or $\mathcal{D}$). The attack outputs a bit, which is **Att** = 0 when it decides that z is a member, and **Att** = 1 when it decides it is a non-member. Two of the most well-known membership inference attacks are the proposals by Shokri et al. [181] and Yeom et al. [205]. The *shadow model attack* by Shokri et al. [181] uses public data to train a set of shadow models, designed to mimic the target model’s functionality. Then, it trains an additional attack model to identify the membership status of a sample using outputs from the shadow models. The *threshold attack* by Yeom et al. [205] assumes that the adversary has access to the loss function of the target model as well as the distribution of the loss on the private training data. Given a data sample z and the model a , the attack queries the model to obtain the loss of z , and decides that z is a member if its loss is below a threshold computed from the distribution information. A weaker variant of this attack assumes the adversary only knows the expected loss of the training set, and uses this value as decision threshold.

A variety of metrics can be used to measure the success of an MIA. The True Positive Rate (TPR) is the probability that the attack correctly identifies a member as such, and the False Positive Rate (FPR) is the probability of incorrectly guessing that a non-member is a member. A popular metric to measure the success of an MIA, which we use in this work, is the *membership advantage*, which Yeom et al. [205] define as $\text{Adv} \doteq \text{TPR} - \text{FPR}$. This metric is 0 when the attack randomly decides the membership of z , and is 1 when the attack always guesses the membership of z correctly.

6.2.2 Differential Privacy in Machine Learning

Differential Privacy (DP), a privacy notion introduced by Dwork et al. [69], has become the gold standard in database privacy and is a popular privacy notion in machine learning:

Definition 6.2.1 ((ϵ, δ) -DP). A training algorithm A provides (ϵ, δ) -DP iff, for any two neighbouring datasets $S, S' \in \mathcal{Z}^n$ (i.e., S and S' differ by a single entry), and all possible subsets of the space of trained models $\mathcal{R} \subseteq \mathcal{A}$,

$$\Pr(A(S) \in \mathcal{R}) \leq \Pr(A(S') \in \mathcal{R}) \cdot e^\epsilon + \delta. \quad (6.1)$$

The parameter ϵ captures the degree of leakage of the mechanism A [69]. Small values of ϵ indicate that a model trained with S is indistinguishable from a model trained with S' which, intuitively, makes it difficult to infer whether or not an element z is in the training set. The parameter δ makes it easier to satisfy the DP constraint by allowing a small chance of failure in the privacy guarantee. It is typical to choose $\delta < 1/n$, where n is the number of elements in the dataset [70].

There are different approaches to developing a differentially private training algorithm A . In the case of neural networks, the differentially private stochastic gradient descent technique by Abadi et al. [11] is widely used. This technique involves clipping the gradients used for updating the network’s weights during training time, and adding Gaussian noise to the average of the gradients.

6.3 Membership Experiments

Evaluating the performance of MIAs is crucial to understanding how dangerous these attacks are in practice and how to protect against them. However, since there are no publicly known cases of MIAs against ML models in practice, researchers typically rely on *membership experiments* to evaluate MIAs. A membership experiment is a theoretical game between the data owner and the adversary, where the adversary tries to guess the membership of a data sample. The game specifies how the data owner generates the training data and which information is available to the attacker. A membership experiment provides a *theoretical definition of membership inference*, specifying the rules one has to follow when evaluating MIAs empirically, and allowing researchers to prove DP-based theoretical guarantees that hold within this controlled environment.

In this section, we review two membership experiments that appear in related work [162, 205]. We argue that these experiments make unrealistic assumptions and thus might not

be representative of what one could expect in a real-world attack. We then propose a new membership experiment that relaxes these assumptions. We call this experiment the Mixture Model (MM) membership experiment. We argue that the MM experiment better represents an MIA that could occur in practice, and thus it provides a more realistic definition of what membership inference is.

6.3.1 Strong-Adversary Membership Experiment

Differential privacy implies, by Definition 6.2.1, that an adversary observing a model a cannot easily distinguish whether it has been trained with $S = \tilde{S} \cup \{z\}$ or $S' = \tilde{S} \cup \{z'\}$, where $\tilde{S} \in \mathcal{Z}^{n-1}$, and $z, z' \in \mathcal{Z}$. This naturally leads to the *strong-adversary membership experiment* that we describe in Algorithm 6 (Exp_{STR}). This experiment has five inputs: an attack Att , a training algorithm A , a set of $n - 1$ samples $\tilde{S}$, and two data samples z and z' . The data owner flips a bit b to decide whether to train a model with $S = \tilde{S} \cup \{z\}$ ($b = 0$) or with $S' = \tilde{S} \cup \{z'\}$ ($b = 1$). The adversary receives the trained model a , the set $\tilde{S}$, and data samples z and z' , and also knows the training algorithm A . The adversary succeeds ($\text{Exp}_{\text{STR}} = 1$) if it correctly learns whether z or z' was a member of the training set of a (i.e., bit b). Nasr et al. [162] use this membership experiment, formulated as a privacy game, to prove that the ϵ -DP guarantees of current training algorithms are tight.¹

Algorithm 6 Strong-Adversary Membership Experiment [162]

```

1: procedure  $\text{Exp}_{\text{STR}}(\text{Att}, A, \tilde{S}, z, z')$ 
2:   Choose  $b \sim \{0, 1\}$  uniformly at random;
3:   if  $b = 0$  then
4:     Train  $a = A(\tilde{S} \cup \{z\})$ ;
5:   else
6:     Train  $a = A(\tilde{S} \cup \{z'\})$ ;
7:   Return 1 if  $\text{Att}(a, z, z', \tilde{S}, A) = b$ ; else 0.
```

The membership advantage in Exp_{STR} is

$$\text{Adv}(\text{Att}, A, \tilde{S}, z, z') = \Pr(\text{Att} = 0 | b = 0) - \Pr(\text{Att} = 0 | b = 1).$$

¹Nasr et al. [162] use the so-called *unbounded* DP notion, where one of the neighbouring datasets has one additional element (i.e., S vs. $S' = S \cup \{z\}$). We stick to the so-called *bounded* DP notion (S and S' have the same size) here since this is the privacy notion followed by Yeom et al. in their membership experiment [205]

Here, the positive summand is the TPR and the negative one is the FPR (if we consider that $b = 0$ is a positive). Note that these terms also depend on the input variables $(\mathbf{Att}, A, \tilde{S}, z, z')$ but we have omitted this dependence in the notation for simplicity. Next, we prove that an (ϵ, δ) -DP training algorithm provides the following upper bound on the membership advantage:

Theorem 6.3.1 (New Membership Advantage Bound). Let A be an (ϵ, δ) -DP training algorithm. Then, for all attacks $\mathbf{Att}$, sets $\tilde{S}$, and data samples z and z' , the membership advantage in Exp_{STR} satisfies

$$\text{Adv}(\mathbf{Att}, A, \tilde{S}, z, z') \leq (e^\epsilon - 1 + 2\delta)/(e^\epsilon + 1). \quad (6.2)$$

Proof. Kairouz et al. [115] show that, for an adversary that wants to distinguish between two neighbouring inputs of a DP mechanism, the following bounds on the adversary's TPR and FPR hold:

$$\text{FPR} + e^\epsilon \cdot (1 - \text{TPR}) \geq 1 - \delta, \quad (1 - \text{TPR}) + e^\epsilon \cdot \text{FPR} \geq 1 - \delta.$$

These expressions follow from (6.1) and trivially apply to the TPR and FPR in Exp_{STR} . Adding these expressions together we get

$$(1 - \text{TPR} + \text{FPR})(1 + e^\epsilon) \geq 2(1 - \delta).$$

Applying the definition of membership advantage ($\text{Adv} \doteq \text{TPR} - \text{FPR}$) and leaving Adv in one side of the inequality yields the desired upper bound. $\square$

For values of δ and ϵ close to zero, the bound above is close to 0. This means that DP is potentially a very strong privacy guarantee since it ensures that the advantage of an adversary that *knows all training set samples except one* (i.e., $\tilde{S}$), and has two possible candidates for the unknown sample (z and z') will always be below a certain value. However, as Nasr et al. [162] admit, this adversary is often unrealistically strong, and perhaps assessing the performance of MIAs that are given all training set samples but one is not very helpful towards understanding MIA in practical cases.

Another problem with this experiment is the *relation* between the membership bit b , the observed samples z and z' , and the trained model a . In Exp_{STR} , the bit b determines the training set of a and thus they are causally *related* ($\Pr(a|b) \neq \Pr(a)$). We argue that there is no such dependence in an actual MIA scenario. In practice, the data owner trains their model once, with their (one and only) training set S . Then, the adversary receives the model and tries to determine whether a sample z is a member of its training set. Here, the bit b is a *property of* z representing its membership status, and not a property of the model a , i.e., $\Pr(a|b) = \Pr(a)$. Yeom et al.'s experiment [205], described next, captures this relation.

6.3.2 IID Membership Experiment

Yeom et al. [205] propose the membership experiment that we show in Algorithm 7. This experiment does not receive the training set S as an input, but a distribution $\mathcal{D}$ from which the samples of S are independently sampled (IID). The data owner trains a using S and then chooses a bit $b \in \{0, 1\}$ uniformly at random. This bit determines whether the adversary receives a member (z chosen randomly from S) or a non-member (a fresh sample $z \sim \mathcal{D}$). The adversary receives the element z and the trained model a , and knows the training set size n , the training algorithm A , and the element distribution $\mathcal{D}$. With this information, the adversary carries out an attack $\text{Att}(z, a, n, A, \mathcal{D})$ that outputs a bit, indicating the predicted membership of z . The adversary succeeds (denoted $\text{Exp}_{\text{IID}} = 1$) if the attack correctly infers the membership status of z .

Algorithm 7 IID Membership Experiment [205]

```

1: procedure  $\text{Exp}_{\text{IID}}(\text{Att}, A, n, \mathcal{D})$ 
2:   Sample  $S \sim \mathcal{D}^n$ ;
3:   Train  $a = A(S)$ ;
4:   Choose  $b \sim \{0, 1\}$  uniformly at random;
5:   if  $b = 0$  then
6:     Draw  $z \sim S$ ;
7:   else
8:     Draw  $z \sim \mathcal{D}$ ;
9:   Return 1 if  $\text{Att}(z, a, n, A, \mathcal{D}) = b$ ; else 0.
```

Yeom et al. [205] prove the following upper bound on the membership advantage in Exp_{IID} :

Theorem 6.3.2 (Yeom et al.’s bound [205]). Let A be an $(\epsilon, 0)$ -DP learning algorithm. Then, for all attacks Att , training set sizes n , and data point distributions $\mathcal{D}$, the membership advantage in Exp_{IID} satisfies

$$\text{Adv}(\text{Att}, A, n, \mathcal{D}) \leq e^\epsilon - 1. \quad (6.3)$$

Note that the advantage is also upper bounded by 1, so this bound is loose for $\epsilon > \ln 2$. More recently, Erlingsson et al. [74] provide a tighter bound using results from Hall et al. [95] for the more generic setting of (ϵ, δ) -DP:

Theorem 6.3.3 (Erlingsson et al.’s bound [74]). Let A be an (ϵ, δ) -DP learning algorithm. Then, for all attacks Att , training set sizes n , and data point distributions $\mathcal{D}$, the membership advantage in Exp_{IID} satisfies

$$\text{Adv}(\text{Att}, A, n, \mathcal{D}) \leq 1 - e^{-\epsilon}(1 - \delta). \quad (6.4)$$

We can see that this bound is less than or equal to 1.

In the full paper [106, Appendix A], we prove that the advantage bound we derive in Theorem 6.3.1 holds in the case where members and non-members are *statistically exchangeable*, i.e., when the joint distribution of a sequence of n members and one non-member does not depend on the order they appear in the sequence. We also explain the IID experiment is a particular case of statistical exchangeability, and thus our bound holds in this case:

Theorem 6.3.4. [New Membership Advantage Bound] Let A be an (ϵ, δ) -DP learning algorithm. Then, for all attacks Att , training set sizes n , and data point distributions $\mathcal{D}$, the membership advantage in Exp_{IID} satisfies

$$\text{Adv}(\text{Att}, A, n, \mathcal{D}) \leq (e^\epsilon - 1 + 2\delta)/(e^\epsilon + 1). \quad (6.5)$$

This bound improves over previous results [74, 205]; i.e., it is tighter than (6.3) and (6.4) for all $\epsilon \geq 0$ and $0 \leq \delta \leq 1$. We prove this in the full paper [106, Appendix B]. Figure 6.1 shows a comparison between the existing membership advantage upper bounds (Yeom et al. (6.3) and Erlingsson et al. (6.4)) and our bound (6.5). We used a value of $\delta = 10^{-5}$ for (6.4) and (6.5), since this is the value of δ we use in our experiments (described in Section 6.4.1) to ensure that δ is less than the inverse of the training set size. Our bound improves Erlingsson et al.’s by almost 0.2 in advantage for relevant privacy values $0.1 \leq \epsilon \leq 2$. These bounds suggest that high privacy settings $0.01 \leq \epsilon \leq 0.1$ completely thwart membership inference attacks (in the IID setting), values of $\epsilon \approx 1$ achieve intermediate privacy levels, and large values of $\epsilon \geq 10$ provide no worst-case privacy guarantee on the membership advantage.

We note that we have considered the original experiment by Yeom et al. [205], where members and non-members are sampled independently from $\mathcal{D}$. In practice, members and non-members are typically different samples, so *sampling without replacement* is a more realistic setting. Sampling without replacement introduces dependencies among members and non-members, but the exchangeability property still holds in that case, and thus our bound still applies. For simplicity, in the remainder of the paper we use the term *IID sampling* to denote either *sampling with or without replacement*, since our findings apply

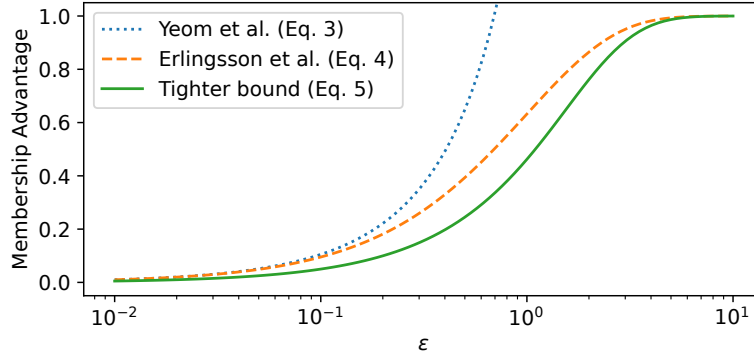


Figure 6.1: Upper bounds on the membership advantage in the IID Membership Experiment.

to both settings. The IID membership experiment captures the evaluation approach of most related works on membership inference [110–112, 162, 181, 205]. These works build the member and non-member sets by randomly splitting a dataset, which simulates IID sampling. However, this procedure excludes the case where there are dependencies between training set samples (e.g., a training set bias). There is a large body of literature in ML that studies this scenario with the goal of improving the accuracy of models on out-of-distribution samples [18, 116, 174]. For example, in a health care setting such as MRI scanning, the images seen in the wild are often from a different distribution to the training data of the model [156]. Spurious correlations in ML datasets have also received a lot of attention in the ML literature as they can have a detrimental effect on model accuracy [157, 177, 178]. Despite the problems of non-IID data being known to the ML community, this case has been completely neglected in the MIA literature.

6.3.3 A Membership Experiment with Data Dependencies

We have argued that the strong-adversary membership experiment considers an unrealistically strong adversary and does not capture the idea that membership should be a property of a given data point, and not the model itself. Also, the IID experiment is limited to a scenario that, as suggested by related work in ML, does not hold in many use cases. Motivated by this, we propose a new membership experiment that extends Exp_{IID} to cases where the training set samples may have dependencies. We consider the particular case where the joint dataset distribution, denoted $\mathcal{D}$, is a *multivariate mixture model*. A mixture model is a probabilistic model that characterizes the distribution of an overall population by taking into account the presence of subpopulations within the overall population. Each of

the subpopulations is characterized by a probability distribution called a *mixture component*. To draw one sample from a mixture model, one first selects one of the mixture components (subpopulations) at random, and then draws a sample from the mixture component. Note that the mixture components may have overlapping support, which implies that, given a point sampled from the mixture model (i.e., the overall population), it might not be possible to identify from which of the mixture components (i.e., subpopulations) it was sampled from. We use *multivariate* mixture components, i.e., each component models the distribution of all attributes of a data sample within a subpopulation. This particular case of $\mathcal{D}$ is amenable to theoretical analyses (e.g., Sect. 6.3.3) and is general enough to accommodate many real-life examples, as we show in Section 6.4.4. We call this the Mixture Model (MM) membership experiment, and denote it by Exp_{MM} .

Algorithm 8 Mixture Model (MM) Membership Experiment

1: procedure $\text{Exp}_{\text{MM}}(\text{Att}, A, n, \mathcal{D})$ 2: Choose $k \sim [K]$, sample $S \sim \mathcal{D}_k^n$; 3: Train $a = A(S)$; 4: Choose $b \sim \{0, 1\}$ uniformly at random; 5: if $b = 0$ then 6: Draw $z \sim S$; 7: else 8: Choose $k' \sim [K] \setminus k$, draw $z \sim \mathcal{D}_{k'}$; 9: Return 1 if $\text{Att}(z, a, n, A, \mathcal{D}) = b$; else 0.	$\triangleright \mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_K\}$
--	--

Algorithm 8 shows the MM membership experiment. The joint distribution $\mathcal{D}$ is a multivariate mixture model with K mixture components. Each mixture component $k \in [K]$ models the training set samples independently following a distribution $\mathcal{D}_k$. We overload the notation of $\mathcal{D}$ in this case for notational simplicity, and use it to denote the set of all mixture distributions $\mathcal{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_K\}$. In short, to sample $S \sim \mathcal{D}$, we choose a mixture component uniformly at random ($k \sim [K]$) and then draw n samples *independently* from $\mathcal{D}_k$ (line 2). Note that, even though the samples are independently drawn from a mixture component $\mathcal{D}_k$, they are statistically dependent since they are drawn from the *same* (unknown) mixture component. When $b = 1$, the non-member z is sampled from a mixture component that is *different* from the training set's component; i.e., $z \sim \mathcal{D}_{k'}$, for $k' \neq k$ (line 8). This experiment can also encompass the IID case in Exp_{IID} when all $\mathcal{D}_k$ (for $k \in [K]$) are identical.

The MM membership experiment accommodates many realistic settings. As an example, consider a case where there are K hospitals in a city. Let $\mathcal{D}_k$ be the distribution that

models the samples from the k th hospital, for all $k \in [K]$. In this example, Exp_{MM} represents the case where the target model a is trained with samples belonging to a particular hospital $k \in [K]$, and the samples from all the other hospitals $k' \in [K] \setminus k$ are non-members. If the distributions of samples in each hospital are slightly different, there will be certain dependencies among members and non-members. Previous theoretical experiments (Exp_{STR} and Exp_{IID}) cannot account for these dependencies. In Section 6.4 we study this and other realistic examples that follow Exp_{MM} .

We note that, following Exp_{MM} , the adversary has statistical knowledge about the possible dependencies between training set samples ($\mathcal{D}$). However, since the choice of k is not revealed, the adversary cannot tell which mixture component distribution the members follow. Even though this statistical knowledge is powerful, it does not provide the adversary any a-priori membership advantage. Indeed, note that, when $\epsilon = 0$ (or, alternatively, if the adversary did not receive the target model a), the adversary cannot do better than randomly guessing the membership of z ($\text{Adv} = 0$). This is because, without the model a , the distribution of z is the same regardless of the membership bit b . However, when $\epsilon > 0$, the adversary can exploit the statistical dependencies among samples to improve the membership inference. We also note that, in our evaluation in Section 6.4, we evaluate off-the-shelf MIAs that *do not use the statistical dependencies* captured by $\mathcal{D}$. However, we will see that these dependencies still cause these existing attacks to perform better than in the IID scenario.

The MM membership experiment captures the relation between b , z , and a , considers a realistic adversary that only has statistical knowledge about the dataset, and accommodates both IID training set distributions ($\mathcal{D}_k \equiv \mathcal{D}_{k'}, \forall k, k' \in [K]$) as well as those that incorporate dependencies among samples or bias ($\mathcal{D}_k \not\equiv \mathcal{D}_{k'}$). Therefore, we believe that this experiment provides a better template to define membership inference, and that evaluating MIAs under this experiment will provide results that are more representative than previous experiments of what one could expect in practice.

DP-based bounds on the generalized membership experiment

We note that existing DP-based bounds [74, 205] on the membership advantage Adv , as well as our new bound in Eq. (6.5), do not apply in the MM membership experiment. We note that this is not a flaw of DP: the training algorithms we evaluated provide differential privacy, but this is not enough to bound the leakage when defining membership inference as in Exp_{MM} . The reason for this is that DP mechanisms limit the effect that a *single* sample has on the output (see Def. 6.2.1). However, when training samples have dependencies,

limiting the effect of a single input is less effective since the remaining data points can also leak information about this input due to their co-dependencies [136].

We provide more intuition as to why the previous bounds do not hold with an example. Without loss of generality and for simplicity, in this analysis we assume that the space of samples $\mathcal{Z}$ is discrete. In both Exp_{IID} and Exp_{MM} , the adversary observes the released model a and a sample z (along with general parameters $n, A, \mathcal{D}$, that we omit from the probability expressions here for notational simplicity). The joint distribution of a and z depends on whether z was a member of a 's training set ($b = 0$), or a non-member ($b = 1$). Next, we write general expressions for the likelihood of a and z given the value of b . We use these expressions to compute the maximum likelihood ratio, which bounds the adversary advantage, and study how an $(\epsilon, 0)$ -DP mechanism bounds this ratio in the IID and non-IID cases.

First, when z is a member ($b = 0$), and using $\tilde{S}$ to denote all other $n - 1$ members, we can write

$$\Pr(a, z|b = 0) = \Pr(z) \cdot \sum_{\tilde{S}} \Pr(\tilde{S}|z) \Pr(A(\tilde{S} \cup \{z\}) = a).$$

Here, $\Pr(\tilde{S}|z)$ is the probability of $n - 1$ training set samples (all but z) conditioned on the fact that z is also a training set sample.

When z is a non-member (denoted $b = 1$), $\tilde{S}$ still denotes $n - 1$ members, and we use z' to define the n th member. Then, using the law of total probability we can write the likelihood as

$$\Pr(a, z|b = 1) = \Pr(z) \sum_{z'} \Pr(z') \sum_{\tilde{S}} \Pr(\tilde{S}|z') \Pr(A(\tilde{S} \cup \{z'\}) = a).$$

(As above, $\Pr(\tilde{S}|z')$ is the probability of the $n - 1$ training set samples $\tilde{S}$ given the remaining training sample z' .)

The maximum likelihood ratio $L_{\max} = \max_{a,z,b} \frac{\Pr(a,z|b=0)}{\Pr(a,z|b=1-b)}$ can be used to bound the adversary advantage.² We show why using an $(\epsilon, 0)$ -DP training algorithm provides a strong bound on this ratio in the IID case (Exp_{IID}), but there are non-IID cases (Exp_{MM}) where the bound is not meaningful. First, recall that an $(\epsilon, 0)$ -DP training algorithm ensures that, for two training sets S and S' that differ in *one* sample, and for all models $a \in \mathcal{A}$, $\Pr(A(S) = a) \leq e^\epsilon \cdot \Pr(A(S') = a)$. In the IID scenario (Exp_{IID}), samples are independent

²We omit the derivations for space issues; in short, $L_{\max}$ lower-bounds the adversary's probability of error by $1/(1 + L_{\max})$, which in turn upper-bounds the advantage by $\text{Adv} \leq (L_{\max} - 1)/(L_{\max} + 1)$. Note this matches (6.5) for $\delta = 0$.

and therefore $\Pr(\tilde{S}|z) = \Pr(\tilde{S})$. The likelihood ratio between hypotheses $b = 1$ and $b = 0$ is bounded by:

$$\begin{aligned} \frac{\Pr(a, z|b=1)}{\Pr(a, z|b=0)} &= \frac{\sum_{z'} \Pr(z') \sum_{\tilde{S}} \Pr(\tilde{S}) \Pr(A(\tilde{S} \cup \{z'\}) = a)}{\sum_{\tilde{S}} \Pr(\tilde{S}) \Pr(A(\tilde{S} \cup \{z\}) = a)} \\ &\leq \frac{\sum_{z'} \Pr(z') \sum_{\tilde{S}} \Pr(\tilde{S}) \Pr(A(\tilde{S} \cup \{z\}) = a) e^\epsilon}{\sum_{\tilde{S}} \Pr(\tilde{S}) \Pr(A(\tilde{S} \cup \{z\}) = a)} \\ &= \sum_{z'} \Pr(z') \cdot e^\epsilon = e^\epsilon. \end{aligned}$$

The same applies to the reciprocal $\Pr(a, z|b=0)/\Pr(a, z|b=1)$. In the MM membership experiment, if we are explicitly in a non-IID case, then $\Pr(\tilde{S}|z) \neq \Pr(\tilde{S})$ and the derivations above cannot be carried out. As an example, consider the pathological case where the mixture components in $\mathcal{D}$ are different and deterministic, i.e., all the training set samples are identical. In that case, $\Pr(\tilde{S}|z)$ is only non-zero when $\tilde{S} = \{z\}^{n-1}$ (in that case, $\Pr(\{z\}^{n-1}|z) = 1$). Then,

$$\begin{aligned} \frac{\Pr(a, z|b=1)}{\Pr(a, z|b=0)} &= \frac{\sum_{z'} \Pr(z') \sum_{\tilde{S}} \Pr(\tilde{S}|z') \Pr(A(\tilde{S} \cup \{z'\}) = a)}{\sum_{\tilde{S}} \Pr(\tilde{S}|z) \Pr(A(\tilde{S} \cup \{z\}) = a)} \\ &= \sum_{z'} \Pr(z') \frac{\Pr(A(\{z'\}^n) = a)}{\Pr(A(\{z\}^n) = a)}. \end{aligned}$$

We can apply the DP definition n times and bound the expression above by $e^{n \cdot \epsilon}$, but this is certainly not useful since n is typically very large. More generally, bounding the expression above requires using DP to provide *group privacy*, which does not yield useful bounds for large n [70]. Even though this is an extreme example, it illustrates why the previous DP-based bounds (that are independent of dataset size) on the membership advantage do not apply to Exp_{MM} .

The pathological example above shows that an upper bound on Adv can be very large in non-IID scenarios. However, there might be particular distributions $\mathcal{D}$ for which one could prove an upper bound that lies closer to the IID bound (6.5). Also, empirical MIAs have been shown to achieve advantage levels Adv far below DP-based upper bounds [111] in the IID scenario. Thus, it is important to empirically measure the performance of MIAs

Table 6.2: Evaluation Setup Summary

Section	Dataset and Experiment	Features	Classes	Members	Non Members	Shadow data
6.4.2	<code>adult</code> cluster	104	2	10 000	10 000	15 132
	<code>compas</code> cluster	15	2	2 000	2 000	-
6.4.3	<code>adult</code> education	87	2	10 000	10 000	10 772
	<code>adult</code> gender	103	2	10 000	10 000	11 192
6.4.4	<code>heart</code> , Cleveland	22	2	303	617	-
	<code>heart</code> , Hungary	22	2	294	626	-
	<code>heart</code> , Switzerland	22	2	123	797	-
	<code>heart</code> , VA Long Beach	22	2	200	720	-
	<code>students</code> , Gabriel Pereira	42	2	423	226	-
	<code>texas</code> , region #3	269	100	10 000	10 000	20 000
	<code>census</code> and <code>adult</code>	100	2	10 000	10 000	20 000

in the non-IID case, as performance could be similar to the IID case. We study this in the next section.

6.4 Empirical Evaluation of Membership Inference with Data Dependencies

In this section, we empirically investigate the effects of non-IID training sets on the performance of off-the-shelf MIAs, following our MM membership experiment Exp_{MM} .

6.4.1 Evaluation Setup

We explain how we instantiate Exp_{MM} with real-world datasets. In each of our experiments, we initialize K disjoint sets of samples $\{D_k\}_{k \in [K]}$ with real data (in most of our experiments, we use $K = 2$). Each of these sets characterizes one of the mixture components: drawing a sample from D_k is simply drawing a sample from the set D_k . We use sampling *without replacement* to ensure that every member and non-member is a distinct data sample. Even though this is an obvious assumption in practice (duplicate samples do not make sense in most problems), we note that all the theoretical experiments in Section 6.3 allow for identical members and non-members.

In each of the sections below, we follow a different approach to build the sets of samples $\{D_k\}_{k \in [K]}$, to study different types of data dependencies. In each experiment, given a training set size n , we take n samples from a mixture component D_k as the member set, and draw n samples at random from all other mixture components as non-members. We train the model on the members, and evaluate each of the attacks we consider on all members and non-members. We compute an attack’s TPR by looking at the proportion of members correctly identified as such, and the FPR by computing the proportion of non-members wrongly classified as members by the attack. Then, the attack’s membership advantage is simply $\text{Adv} = \text{TPR} - \text{FPR}$. We label this advantage as “non-IID” in our plots. We also measure the classification accuracy of the model over members and non-members in this non-IID case, to give insight into the attacks’ performance.

In order to study the effect of data dependencies vs. independent data sampling, we also measure the membership advantage if members and non-members were IID samples. To do this, for each non-IID experiment, we merge the member and non-member sets, and generate a new member and non-member set by resampling from this merged set (we keep the original member and non-member set sizes). The advantage in this case represents the attack’s performance in the IID experiment, and we label it “IID” in our plots.

Datasets We use three publicly available datasets considered in prior work [124, 181] and three additional datasets from the UCI machine learning repository [64]:

1. The `adult` dataset [122], which contains 48 842 data samples from the 1994 US census, each with 14 attributes such as age, gender, and education. The binary classification task is deciding whether the individual earns more than \$50k a year.
2. The `compas` dataset [15], extracted from ProPublica’s investigation into racial bias in ML, which contains 6 172 samples with 15 attributes such as age, sex, and number of prior offenses. The classification task is predicting whether or not an individual re-offended within 2 years.
3. The `texas` dataset [4] contains a series of records of inpatient stays in various hospitals published by the Texas Department of State Health Services. The classification task is predicting the procedure. We follow the approach by Shokri et al. [181] and compute the top-100 most popular procedures as the classification label (we discard any rows not in the top-100). The final dataset contains 350 280 samples each with 66 attributes such as length of stay, age, and total charges.

4. The **heart** datasets [5] are four datasets collected from hospitals in Cleveland, Hungary, Switzerland, and the VA Long Beach. The datasets contain 303, 294, 123, and 200 data samples, respectively. We use the 14 attributes common to all datasets such as age, cholesterol, and fasting blood sugar with the classification task of predicting if the patient has a heart condition.
5. The **students** dataset [46] contains data samples from a study on student achievement in Portuguese schools. We use the Portuguese language dataset, which has 649 samples from two schools (423 and 226 samples each) containing 30 attributes such as age, study time, and number of absences. The classification task is predicting if the student passed or failed the course, based on the final grade. We remove the two intermediate grades from the attribute list, as they are highly correlated with the final grade.
6. The **census** dataset [6] contains data extracted from the 1994 and 1995 US Census. We take all 99 827 data samples from the 1994 dataset and extract the 10 attributes in common with the **adult** dataset. The classification task is to predict whether an individual earns more than \$50k a year.

We replace missing values with the mean or mode value for that attribute and use a one-hot encoding for all categorical attributes. For example, if the categorical attribute has values red, green, and blue we add three new binary attributes, one for each colour. Since we only used a subset of the attributes for certain datasets, the rows were not always unique. To address this, we remove duplicate rows, keeping a single copy of each duplicate. Table 6.2 summarizes how we use these datasets in our experiments. In most experiments, we set a value for n ($n = 10\,000$ in most cases, and $n = 2\,000$ in **compas**), and sample n members and n non-members using the procedure described above. When the remaining data is large enough, we use it to evaluate the shadow model attack (when we do not have enough data, denoted “–” in Table 6.2, we skip this attack). In experiments where the datasets are small (**heart** and **students**), we use an uneven number of members and non-members (we explain this in Section 6.4.4).

Model Architecture Following the work of Jayaraman et al. [111], and others [11, 181], we use a ReLU network with 2 hidden layers, each with 256 neurons trained for 100 epochs as the target model in all evaluations. The models use the DP ADAM Gaussian Optimizer and RDP accountant from TensorFlow Privacy [7] with ℓ_2 regularization to avoid overfitting. The default hyper-parameters are 10^{-5} as the ℓ_2 regularization coefficient, 10^{-2} as the learning rate, and 200 as the batch size [111]. We vary ϵ in our evaluations and fix δ to

equal 10^{-5} . This follows the general recommendation that δ should be less than the inverse of the training set size [70]. We note that $\epsilon = 0$ still leaks information, since $\delta > 0$.

Membership Inference Attacks We consider two membership inference attacks evaluated in prior work [111] that we summarized in Section 6.2.1. The *shadow model attack* [181] requires a large pool of publicly available data in order to train shadow models. Therefore we only run this attack on datasets that have leftover samples after building our member and non-member sets (see Table 6.2). Following prior work [111, 181], we train five shadow models, each with the same architecture as the target model. The attack model uses the same architecture as the target model but with 64 neurons in the hidden layer. We consider two variants of the *threshold attack* [205]: one that knows the true loss distribution and therefore chooses the optimal decision threshold (optimal threshold attack) and one that uses the average training loss as decision threshold (average threshold attack). We remark that we do not modify these attacks. The attacks only get black box query access to the model a (with confidence scores), the training loss (log loss attack), public data following $\mathcal{D}$ (shadow model attack), and the sample z . That is, they do not use all the information that we allow the attacker in Algorithm 8. Specifically, we omit the set size n and the training algorithm A . The attacks were not designed to exploit dependencies between training samples and the adversary never explicitly learns these dependencies. Furthermore, in each experiment, we are careful not to include attributes that can trivially identify the mixture component that a sample was drawn from (we clarify this below).

Implementation We used Python 3.8 for our implementation³, building upon the code of Jayaraman et al. [8]. This code includes the implementation of the threshold attack and shadow model attack, where the latter is based on Shokri et al. [181]. To implement RDP, we use the RDP accountant in TensorFlow Privacy.

6.4.2 Cluster-based Dependencies

We begin by studying to what extent data dependencies can affect MIAs. To do this, we artificially create two sets D_1 and D_2 that have samples whose features lie in different regions of the feature space. A similar idea was explored in previous work [157] by adding spurious perturbations to the training data that caused a large discrepancy between training and testing accuracy. Instead of modifying existing attributes, we take a real-world dataset

³Source code available at <https://github.com/t3humphries/non-IID-MIA>

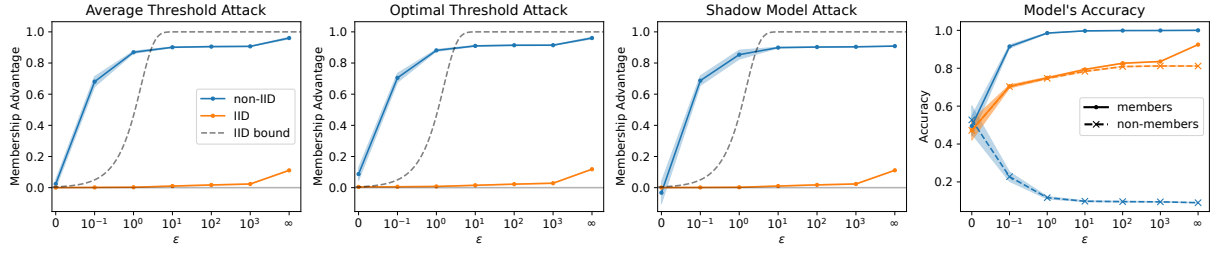


Figure 6.2: Results in **adult** dataset with cluster-based dependencies.

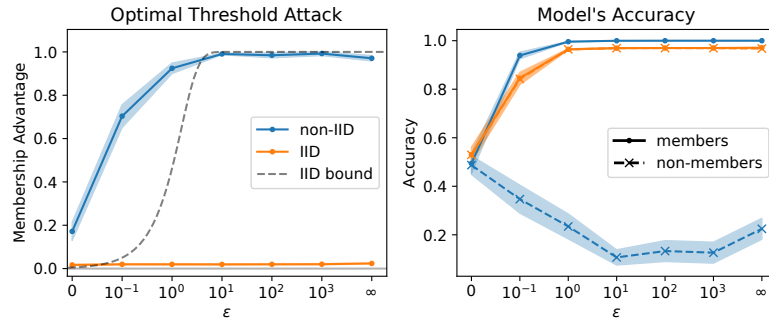


Figure 6.3: Results in **compas** dataset with cluster-based dependencies.

and run a k -means clustering algorithm with $k = 2$ to find clusters of samples that lie in different regions of the feature space. We run the clustering algorithm independently for all the samples of each class, and add all samples in cluster i to the set D_i ($i \in \{1, 2\}$). Intuitively, D_1 and D_2 represent mixture components where the characteristics used to distinguish the classes are significantly different.

We use `adult` with $n = 10\,000$ and `compas` with $n = 2\,000$ in this experiment. We evaluate what happens when we use D_1 to sample the members and D_2 for the non-members, and vice-versa. We use leftover samples in `adult` (up to n per cluster) to train the shadow models for the shadow model attack (see Table 6.2). We do not evaluate the shadow model attack for the `compas` dataset, since we do not have enough samples to provide the adversary.

We run the clustering algorithm once to define D_1 and D_2 , and then repeat the creation of member and non-member sets, the training, and the attack evaluation 50 times. For each of these experiments, we also run the IID counterpart as explained above, i.e., by randomly mixing the member and non-member sets and repeating the training and attack evaluation.

Figure 6.2 shows the average advantage of the average threshold attack, optimal threshold attack, and shadow model attack (shaded areas are the 95% confidence intervals for the mean) versus the privacy level ϵ . We only show the results for the case when members are sampled from D_1 and non-members from D_2 (the results for the opposite scenario were very similar). For the value $\epsilon = \infty$, we simply use non-private SGD. For reference, we also plot our membership advantage bound (6.5), which is only guaranteed to hold in the IID scenario. The rightmost plot includes the average classification accuracy of the target model over the member and non-member sets.

We see that these cluster-based dependencies significantly increase the adversary’s performance compared to the IID case. In general, we observe that the attack performance increases as the privacy level decreases (higher ϵ). For $\epsilon \geq 1$ we see that the attack performs alarmingly well, close to perfect advantage. Our results also confirm that the advantage bound does not hold in non-IID cases.

The classification accuracy gives insight into the performance of the threshold attacks. Recall that these attacks classify all points with logloss *below* a threshold as members. The high discrepancy between the loss of members and non-member samples makes it easier for these attacks to identify their membership correctly. Another interesting observation is that, in Figure 6.2, the classification accuracy remains constant after $\epsilon > 1$ but the non-IID advantage of the threshold attack further increases when $\epsilon = \infty$. This corroborates the observation of Yeom et al. [205] that overfitting is not the only factor at play in the success of the attack.

In Figure 6.3, we show the results of the optimal threshold attack for the `compas` dataset. These results confirm our findings in the `adult` dataset, and even reach an almost-perfect advantage (≈ 1) for $\epsilon = 10$. To summarize, we have shown that, using off-the-shelf attacks and datasets, certain dependencies among members/non-members can significantly increase the performance of MIAs. The advantage we observe not only crosses the DP bound for the IID case, but can reach the maximum value of 1 in the worst case.

6.4.3 Dependencies Caused by an Attribute Bias

We have seen that the cluster-based dependencies can yield extremely high levels of membership advantage. However, a dependence this strong might not occur in practice. In this section, we consider a more realistic and interpretable dependency, based on an attribute in the dataset, that we call an *attribute bias*. We build two experiments using the `adult` dataset: one using the gender attribute (there are only two genders in this dataset, ‘Male’ and ‘Female’), and another one using the education level (we consider that samples with at least ‘some-college’ education are “high education”, and all the others are “low education”). For each experiment, we use member and non-member sets with $n = 10\,000$ samples, and a bias value $p \in [0, 1]$ representing the proportion of the training set that has the particular attribute value.

We generate the sets D_1 and D_2 as follows. We use the gender attribute as an example, but the set generation works similarly for the education attribute. First, we generate D_1 by taking $\lceil p \cdot n \rceil$ samples at random from the dataset with the gender attribute ‘Male’, and the remaining samples have the ‘Female’ attribute. We build D_2 by taking an even number of samples with each gender attribute value (i.e., 5 000 ‘Male’ and 5 000 ‘Female’). We delete the gender attribute from all samples after generating the sets, so that the adversary cannot use the gender directly in identifying members and non-members. As before, we use remaining data samples to train the shadow models for the shadow model attack, ensuring we do not give the adversary more than n samples per attribute value. Since our goal here is to investigate training set bias, we train with the biased set (D_1) and vary the bias level p . The non-members (D_2) have an equal number of samples from each attribute value. We repeat the set generation, training, and evaluation process 50 times for each of the values of the bias p that we test.

Figure 6.4 shows the average advantage of the average threshold attack, optimal threshold attack, and shadow model attack for the gender attribute bias, versus the privacy level ϵ . Each colour represents the performance for a different training set bias, as shown in the legend. We show only some of the values of the bias p for the model’s classification accuracy, for readability.

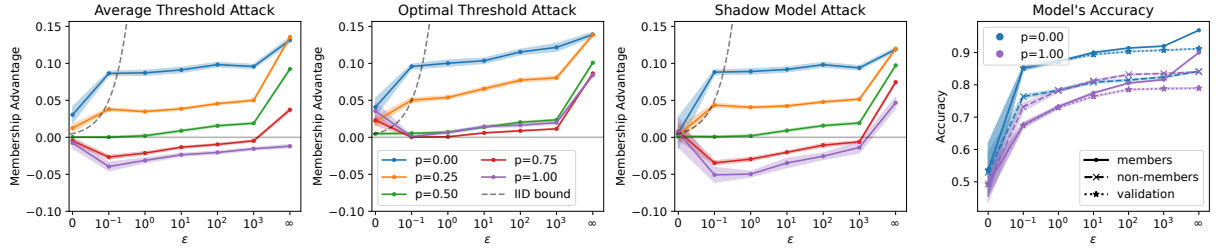


Figure 6.4: Results in `adult` dataset with gender bias (p determines the proportion of males in the member set).

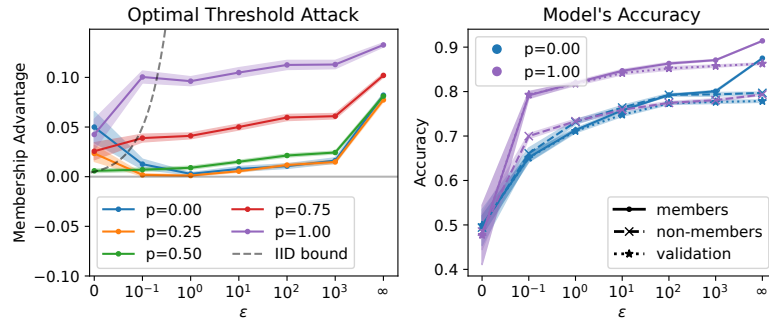


Figure 6.5: Results in `adult` dataset with education bias (p determines the proportion of individuals with low education in the member set).

We note that, when $p = 0.5$, both the member and non-member sets have the same number of samples from each attribute value, so this can be considered an instantiation of the IID experiment. Training bias ($p \neq 0.5$) can significantly increase the membership advantage, and can even have more impact than the privacy level ϵ . In fact, we see that the adversary achieves similar advantage in a model trained on unbiased data (50% males) without privacy ($\epsilon = \infty$), than in a model trained on highly biased data (0% males) with a high privacy level ($\epsilon = 0.1$). For bias levels $p \geq 0.75$, the membership advantage can be negative. The classification accuracy gives insight into this effect for the average threshold attack. When the training set consists of all male samples ($p = 1.00$), we see that the classification accuracy for non-members is higher than for members. This means that the classification task (salary prediction) is likely easier for predominately female samples than is on male samples (even when trained on males only). This misleads the threshold attack (which classifies all samples with low loss as members) causing it to perform poorly (negative Adv) when the training set is male-dominant.

In this experiment, we also consider the classification accuracy of a validation set that has the same bias as the training set. That is, we sample 2500 data points that are disjoint from the training set with the same attribute bias as the training set. We see that the target models show very similar member and validation set accuracy, especially for low values of ϵ . This suggests that, to a data owner who only has access to biased data, the model appears to generalize well. Even if the data owner runs a tool such as ML Privacy Meter [155] to check their model, they might underestimate the vulnerability of their model.

We plot the results using the education split for the optimal threshold attack only in Figure 6.5. The results are qualitatively similar to the ones with the gender split. In this case, predicting the income of individuals with low education levels is generally easier than on individuals with high education. Even when training with high-education samples only ($p = 0$), the model’s accuracy over non-members (samples with low education) is higher than in members, which causes the MIAs to perform poorly for low values of p . However, when p is large (majority of low-education samples in the training set), we see that the MIAs perform better, breaking the DP bound for the IID case.

6.4.4 Inherent Dataset Dependencies

We have seen that dependencies play a significant role in the success of membership inference attacks. A natural question to ask is whether such dependencies occur in practice. While there are no public examples of MIAs in the wild, we study naturally occurring dependencies found in off-the-shelf datasets. We do this by either finding different datasets for a similar

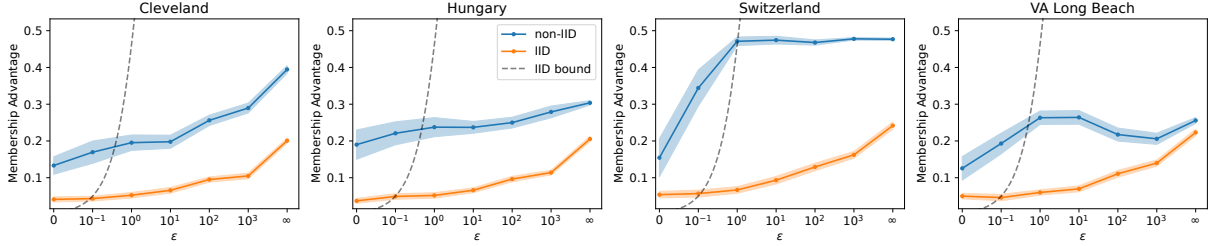


Figure 6.6: Performance of the optimal threshold attack in the **heart** dataset when members belong to a particular database (hospital/institution), and non-members are taken from all other databases.

classification problem or splitting existing datasets based on an attribute that would define a training set in practice, such as the location/institution where the samples were collected.

Hospital data: heart condition detection For our first dataset-based dependency experiment, we use the **heart** datasets, which consist of four different datasets (Cleveland, Hungary, Switzerland, and the VA Long Beach), each containing the samples from one hospital/institution. The classification problem is detecting whether or not a patient has a heart condition (the classification problem is binary). We use all the samples from each database to define the $\{D_i\}_{i \in [4]}$ sets, and evaluate the performance of MIAs when we train with one database and all the samples from the other three databases are non-members.

Figure 6.6 shows the results when training with each of the hospitals. We see that there is certain property present within each hospital, since the membership advantage when we train with one hospital dataset (non-IID) is significantly higher than when we shuffle all datasets (IID). We note that the advantage in the IID case slightly breaks the bound. We posit that this is because the RDP accountant assumes batches are sampled without replacement and each batch is sampled independently. In practice, and in our implementation, batches are non-overlapping. This difference in sampling would be particularly noticeable when the training set is small (as overlaps are more probable), which explains why the advantage in the IID case is slightly above the bound.

School data: students proficiency detection Next, we consider the **students** dataset, which contains data about students from two Portuguese schools (Gabriel Pereira and Mousinho da Silveira), and the classification problem is to predict whether the student passes the course. We use all 423 samples from Gabriel Pereira school (D_1) as the training

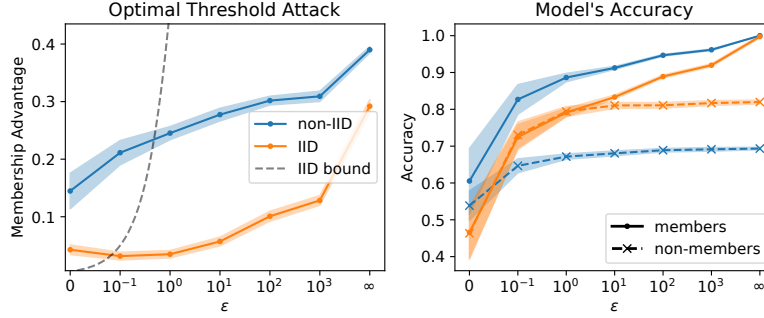


Figure 6.7: Results in `students` dataset when members are from the Gabriel Pereira school, and non-members from Mousinho da Silveira school.

set, and all 226 samples from Mousinho da Silveira school (D_2) as non-members. Figure 6.7 shows the results. As before, we see that there is a common property in the students of each school that helps MIAs succeed with higher probability than in the case where we shuffle the data from both schools (IID).

Hospital data: large dataset with a multiclass problem Next, we use the `texas` dataset, which is a larger dataset (350 280 samples) with the 100-class problem proposed by Shokri et al. [181]. Each sample contains an attribute specifying the health region where it was collected. We assign samples from health region number 3 to the set D_1 , and assign all other samples to D_2 . For each run of this experiment, we randomly sample $n = 10\,000$ samples from D_1 to use as members/training, and use the same number of samples from D_2 as non-members (we restrict the sizes for computational reasons). We give n leftover samples from each set to the adversary to train the shadow model attack. We note that such split could naturally occur in practice (e.g., the local authorities of health region 3 decide to train a machine learning model using data for all the hospitals in that region). Figure 6.8 shows the results. In this case, the difference between the non-IID case (training with health region 3) and IID case (training with samples from all health regions) is smaller than in previous experiments. We can see that the classification accuracy is only slightly higher for members in the non-IID than in the IID case (the opposite is true for non-members), which explains the more modest advantage increase due to the training set dependencies. However, the relative increase in advantage between the IID and non-IID experiments is significant, especially in the threshold attack when $\epsilon \in [1, 100]$.

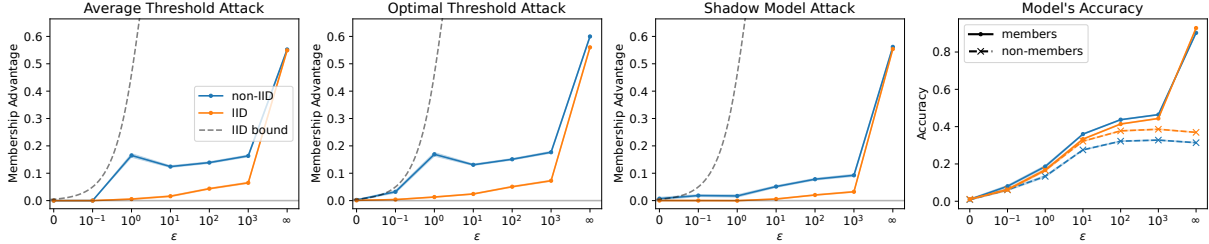


Figure 6.8: Results in `texas` dataset when members are from hospitals in region 3, and non-members are from any other region.

Census data: dataset curation dependencies Finally, we consider the dependencies that stem from techniques a researcher or institution follows when curating a dataset. We use both `adult` and `census` datasets, which have been extracted from *the 1994 US Census data*, but were curated into ML training data by different researchers. Specifically, each research group extracted a different number of records: the `adult` dataset consists of 48 842 records extracted using certain binary rules to filter out records, and the `census` dataset contains 299 285 records, but does not specify any filters. Both datasets consider the classification task of predicting if an individual’s income is above 50 000 USD (binary classification). We take 10 attributes that both datasets have in common. The set D_1 is the data from `census`, and D_2 contains the samples from `adult` (we remove duplicates so that $D_1 \cap D_2 = \emptyset$). As in the previous experiment, we set $n = 10\,000$ and sample n data points from D_1 as members and n from D_2 as non-members.⁴ We further draw n samples from each set to train the shadow model attack. We show the results in Figure 6.9.

We see that, even though the census data comes from the same source, the curation of these datasets introduces dependencies that can certainly help MIAs. The performance across all attacks is considerably higher for the non-IID setting, and DP-SGD does not seem to have a big effect at preventing this leakage. We note that the noise that DP-SGD adds during training decreases with n ; this explains why, in this dataset, even small values of ϵ are not effective at stopping the MIAs in presence of data dependencies.

⁴We remark that, since both of these datasets are extracted from the same source (the US Census), there a possibility that multiple unique records can describe the same individual. However, we consider membership inference at the data point level for this experiment.

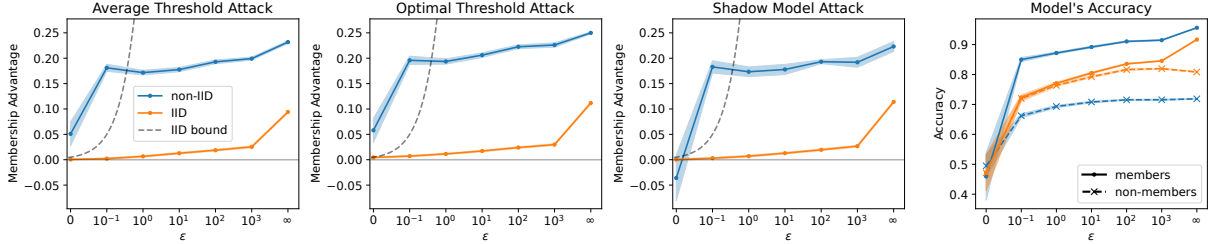


Figure 6.9: Results when training a model with `census`, where non-members are from `adult` dataset.

6.5 Discussion: Connection to Property Inference

In this paper, we investigate the effects of data dependencies on membership inference attacks. We show that *statistical differences* between members and non-members can improve the performance of existing off-the-shelf MIAs. One can interpret these data dependencies as a *property* that allows distinguishing members from non-members. This property can be tangible, such an existing attribute that already distinguishes members from non-members, or a non-interpretable and arbitrarily complex combination of attributes.

Property Inference Attacks (PIAs) [19, 82, 145] aim at revealing a property of the training set that the data owner might have wanted to keep secret. In contrast, recall that MIAs aim to infer whether an individual sample is a member of the training set. Given their definitions, MIAs and PIAs are very different attacks, with different goals altogether. However, by defining the data dependencies among members as a property, one could devise a *new MIA* that first runs a PIA to reveal a property of the training set and then checks the presence/absence of this property in the target sample to decide on its membership. This *MIA-via-PIA* draws connections between membership and property inference in the non-IID case: a successful PIA could lead to a successful MIA. In this section, we first discuss the feasibility of an MIA-via-PIA, then we discuss the privacy implications of this attack and, finally, explain the differences between an MIA-via-PIA and the MIAs we evaluated in Section 6.4.

Feasibility of MIA-via-PIAs The first step in an MIA-via-PIA is finding a property that separates members from non-members. PIAs receive a description of the property to be inferred as an input, e.g., as two complementary hypotheses [19, 82] or as a binary function of the samples' attributes [145]. Generally speaking, finding the right property description that separates members from non-members is challenging. In the theoretical MM

membership experiment (Algorithm 8), the adversary has access to $\mathcal{D}$, a joint distribution made up of mixture components. Thus, in this setting, the adversary has a description of a set of properties $[K]$ representing the various mixture components. The adversary could train a meta-classifier [82] that receives a trained model as input, and infers which mixture component $\mathcal{D}_{\hat{k}}$ ($\hat{k} \in [K]$) the training samples followed. However, in practice, this property inference is likely to be unsuccessful for a number of reasons. First, the assumption that the adversary receives $\mathcal{D}$ is very strong and unlikely to occur in practice. Second, the number of mixture components K is likely very large or might not even be a discrete set (e.g., in the case of hospital data, K could be the total number of “possible hospital distributions” that exist, which might not even be finite). This further increases the difficulty of the property inference (estimation of $\hat{k}$).

The second step in an MIA-via-PIA is to use the property $\hat{k}$ to attempt to infer the membership of a sample. For example, given the target sample z , the attacker could check its likelihood of being a member by checking if $\Pr(z|z \sim \mathcal{D}_{\hat{k}}) > \Pr(z|z \sim \mathcal{D}_{k'})$ for all $k' \neq \hat{k}$. Performing this step can be highly difficult for different reasons. First, we note that MIAs are a *statistical game*: even when the adversary correctly estimates the property, it is not always possible to correctly guess the membership of the target sample.⁵ Second, the mixture components’ support is very likely overlapping such that, given a particular data point z , there might be several mixture components from which the sample might have come from (i.e., $\Pr(z|z \sim \mathcal{D}_{\hat{k}}) \approx \Pr(z|z \sim \mathcal{D}_{k'})$ for at least one $k' \neq \hat{k}$). In these cases, the MIA-via-PIA will likely fail. Note that this will often be the case when K is large, which is likely to happen in practice, as we argue above.

Privacy Violations Next, we discuss the meaning of privacy violations under an MIA-via-PIA. While PIAs violate the privacy of the training set as a whole, MIAs violate the privacy of individual members. This questions whether the privacy violation of a successful MIA-via-PIA is an individual or a group privacy violation. The MIA-via-PIA approach decides that any sample z that has the property is a member, regardless of whether or not that sample was truly in the training set. In that case, it would seem that correctly guessing membership is independent of the actual membership of the sample, and thus is not an individual privacy violation but a group privacy violation. However, we want to reiterate that MIA is a *statistical game*. In situations where there is a property that accurately distinguishes members from non-members, an MIA-via-PIA is a smart strategy from a Bayesian inference point of view, that could increase the adversary’s odds of winning

⁵Note that 100% accuracy in membership inference is not possible, since the number of possible datasets typically exceeds the number of possible model parameters, and thus at least two different datasets may result in the same model, by the pigeonhole principle.

the statistical membership game. Leveraging the property *does not change the meaning* of the privacy violation from an individual to a group violation. We note that leveraging dependencies to improve the accuracy of statistical attacks is common in many privacy areas, such as location privacy (spatio-temporal correlations between a user’s locations can help the adversary’s estimation of these user’s individual locations [182]), database privacy (correlations between client queries aid finding out the underlying secret keyword of each individual query [168]), etc. These privacy violations are generally considered as individual, and not group, violations.

MIAs under data dependencies While we have argued that an MIA-via-PIA is a valid MIA that can be successful, a PIA is not necessary to build a successful MIA under statistical dependencies. Our theoretical experiments follow prior work [205] and consider an informed adversary with knowledge of $\mathcal{D}$. We have shown, theoretically, that DP bounds do not hold against such adversary (Section 6.3.3). However, in practice, assuming the adversary knows $\mathcal{D}$ might not be realistic. Our empirical evaluation considers a particular case of the theoretical membership experiment where the adversary simply does not use (and does not know) $\mathcal{D}$. We evaluate popular off-the-shelf attacks and see that these attacks achieve high accuracy under data dependencies, exceeding the DP bounds that hold for the IID scenario. This shows that knowledge of $\mathcal{D}$ (which could enable a MIA-via-PIA) is not necessary to exploit the data dependencies and break the IID DP bounds.

Regardless of whether one uses an MIA-via-PIA in non-IID settings, it is a fact that data dependencies affect the performance of off-the-shelf MIAs (our empirical evaluation demonstrates this) and thus MIAs in the wild will inevitably (and perhaps unintentionally) also be affected by this. We believe that MIA advantage in the wild will be closer to the experiments that we have seen in this paper, and we maintain that future evaluations of MIAs should consider realistic member/non-member splits, since we have seen that random shuffling underestimates the privacy risks of membership inference. Finding techniques that help mitigate the leakage that stems from data dependencies, which has been ignored by past work, is an interesting topic for future work.

6.6 Related Work

Chatzikokolakis et al. [40] provide a similar theoretical bound for the case of pure differential privacy. However, since DP-SGD uses the moments’ accountant, which gives approximate DP, our bound is more useful in practice. We remark that the goal of this paper is not to

be competitive with state-of-the-art attacks in MIA [33, 112, 140, 176, 184, 198, 203]. Instead, we show the amplification effect of data dependencies on existing, well-known attacks by Shokri et al. [181] and Yeom et al. [205]. Showing that a single MIA breaks the upper bound for a single attack is enough to prove our point.

There have been several works that investigate the performance of MIAs under different assumptions. Nasr et al. [161] provide a comprehensive privacy analysis of MIAs under several white-box conditions and design an inference attack that targets vulnerabilities in the stochastic gradient descent algorithm. Recent work by Jagielski et al. [110] improves MIA success by developing a novel data poisoning attack. Nasr et al. [162] show experimentally that the strong adversary (see Alg. 6) can reach the theoretical DP upper bounds. These works all show stronger attack success than previously reported. However, they achieve this performance by giving the adversary increased (often unrealistic) capabilities while only considering the case of IID data. We instead focus on the weaker black box model of attacker capabilities and vary the data distribution (to be more realistic) to strengthen the attacks.

Kulynych et al. [124] demonstrate that data samples from underrepresented communities are more vulnerable to MIAs. Bagdasaryan et al. [20] find that DP-SGD results in a disproportionate amount of accuracy loss for vulnerable groups. These works complement our findings that MIAs are stronger when the training data has dependencies. Xiong et al. [202] study the risks of non-IID data in the federated learning setting. Hu et al. [100] design a new attack to uncover the source of data points (such as which hospital contributed the data point) in the federated setting. This work complements ours by showing an increase in attack performance when considering clients with different data distributions. Similarly, Gomrokchi et al. [91] study the problem of MIAs against reinforcement learning and find that temporal correlation between training trajectories plays a major role in attack success.

We recall that property inference attacks aim to discover hidden properties in a target classifier’s training dataset, such as the proportion of members in the dataset that are students [19]. It is well known that DP does not protect against property inference attacks [19, 145]. As discussed in Section 6.5, our goal is not to conduct a property inference attack or be competitive with state-of-the-art attacks in this field [19, 82, 145, 170, 209]. Instead, we show the effects of datasets that are non-IID on MIA attacks using properties as one way to introduce a dependency in the datasets.

While our work is the first to investigate the effects of dependent data on MIAs against Neural Networks trained with DP-SGD, other works have studied the limitations of DP under data dependencies, as summarized by Tschantz et al. [194]. Kifer and Machanavajjhala [119] claim that DP mechanisms do not always limit participation inference, and later propose

the Pufferfish framework [120], which they use to show that data correlations can cause additional information leakage compared to the IID case. Li et al. [131] explain that DP implies membership privacy only when data is mutually independent. Liu et al. [136] empirically study the effect of data dependencies on inference attacks against the Laplace mechanism. They show that an adversary with knowledge of these dependencies can violate DP bounds that hold in the IID scenario. Although our conclusions are similar (besides the fact that we work in the ML domain), an important difference between this work and ours is that the attacks we evaluate do not use knowledge about these data dependencies. Instead, we show that existing attacks [181, 205], which were not designed to exploit data dependencies, also benefit from them.

6.7 Conclusion

We show that current MIA experiments and evaluations hinge on the restrictive assumption that members and non-members are IID data samples. We studied the performance of MIAs when there are statistical dependencies among the training set samples, and found that they pose a far greater threat than previously reported. We show that current state-of-the-art MIAs can achieve near-optimal performance when we introduce artificial dataset dependencies, but even naturally occurring dependencies can increase MIA performance compared to the IID scenario. We show theoretically and empirically that, when members and non-members are not IID, the previous theoretical bounds of DP do not apply. Our work demonstrates that data dependencies should be taken into account when studying MIA performance, as they are a realistic assumption that, if ignored, can lead to a significant underestimation of the privacy risk that MIAs pose.

Chapter 7

Conclusion

In response to the research question, we first presented a collection of case studies that highlight the effectiveness of our approach to identify the key bottleneck in private systems and design a solution tailored to the specific context of that system.

In Chapter 2, the key challenge inhibiting deployment was the performance overhead of secure inference, specifically the computation time of non-linear layers. Naively replacing the non-linear layers with polynomial approximations leads to poor inference accuracy or unstable training. Our approach designed a custom ML training pipeline and secure computation protocol that are closely integrated to achieve significant improvements. Our co-design technique significantly reduced the performance trade-off, yielding between 3 and 110 $\times$ speed-ups over the state-of-the-art solutions.

In Chapter 3, the existing approaches suffered from the inherently bad utility of the local model of DP, exacerbated by the correlated relationship of keys and values in key-value data. Our solution utilized the problem context of computing a DP output in the local model to design a secure computation protocol with reduced runtime through additional DP leakage. The result was a system that boasts utility in line with the central model ($\sqrt{m}$ improvement where m is the number of clients), while improving the runtime of the secure computation from the order of days to seconds.

In Chapter 4, the state-of-the-art required adding noise to the cluster centres proportional to the entire private data domain. Our solution took the approach of evolving a coreset of values, so the private data simply has to vote for its best representative. This process of voting significantly reduced the amount of noise, and therefore error, introduced by differential privacy. The result outperforms related work on various benchmark datasets and also includes a high-dimensional variant to remain competitive as datasets scale.

In Chapter 5, we needed to find a way to communicate the error of a private median query without significantly increasing the error of the median, unlike previous work. Our solution was to design a new function to estimate a bound on the error as a post-processing step to an unmodified private median query. By observing the properties of median queries specifically, we were able to significantly reduce the sensitivity of the utility function and gain between 14 and 850% improvement in the error of the median while providing a similarly tight and interpretable error bound.

While the design of each of these case studies was heavily influenced by the deployment bottlenecks of the given systems, we extract the transferable contributions from each case that inspire future solutions.

- Securely evaluating polynomials is a fundamental problem in secure computation (directly occurring in applications like PSI or through approximations of non-linear functions like in this work) for which we provide a lightweight MPC protocol in Chapter 2.
- Selective MPC (Chapter 3) was one of the first papers to use DP leakage to reduce the runtime of securely computing a DP statistic. This approach simply relaxes the existing privacy parameter, rather than a reduction of security guarantees that typically happens in leaky cryptography. This design model provides an interesting way to overcome the utility, performance, and privacy trade-off of the local trust model.
- The unique ability of Selective MPC to capture negative noise through limited views is also widely applicable, as many previous works employ dummy traffic following a DP distribution.
- Our modifications to improve private evolution in Chapter 4 can be applied to the other data modalities such as images and text that are currently deployed by companies like Apple and Microsoft.
- Our post-processing approach (Chapter 5) of estimating the error bound after the statistic (to maintain the quality of the original statistic) can be extended to give bounds for other data-dependent privacy distributions.

In response to the second part of the research question, Chapter 6 provided an example highlighting why it is important to ensure a meaningful privacy definition, by clarifying the difference between two popular notions of privacy in machine learning. Before our work, it was common to oversell the protection of DP in machine learning, even at low levels

of theoretical protection. Our work provides clear evidence to ensure that stakeholders looking to deploy private machine learning are more informed when making privacy claims. An avenue of future work for this project is to design a privacy mechanism that effectively bounds membership inference in the case of data dependencies. At a higher level, future work in auditing should continue to look holistically at the threat models of a system in search for unrealistic assumptions and the effects of relaxing them.

We have outlined some next steps or potential future applications of the specific cases given in this thesis. However, looking at the larger research question of creating deployable private alternatives for data science systems is a truly immense open problem. To this end, we believe the most effective way forward is to continue finding the most influential systems to work on next through discussions with relevant stakeholders. We can then support these stakeholders by introducing designs that ease the contention of privacy trade-offs and providing data-driven evidence for selecting the right privacy definitions. Ultimately, these efforts move us closer to an ideal world where private systems are ubiquitous.

References

- [1] <https://github.com/LucasFenaux/PILLAR-ESPN>.
- [2] <https://github.com/tensorflow/tensorflow>.
- [3] <https://pytorch.org/>.
- [4] <https://www.dshs.texas.gov/THCIC/Hospitals/Download.shtm>.
- [5] <https://archive.ics.uci.edu/ml/datasets/Heart+Disease>.
- [6] [https://archive.ics.uci.edu/ml/datasets/Census-Income+\(KDD\)](https://archive.ics.uci.edu/ml/datasets/Census-Income+(KDD)).
- [7] <https://github.com/tensorflow/privacy>.
- [8] <https://github.com/bargavj/EvaluatingDPML>.
- [9] Josh Aas and Tim Geoghegan. Introducing ISRG Prio Services for Privacy Respecting Metrics. <https://www.abetterinternet.org/post/introducing-prio-services/>, 2020. Accessed 2022-04-18.
- [10] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, pages 308–318, New York, NY, USA, October 2016. Association for Computing Machinery.
- [11] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 308–318, 2016.

- [12] John M. Abowd. The u.s. census bureau adopts differential privacy. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 2867, New York, NY, USA, 2018. Association for Computing Machinery.
- [13] Abbas Acar, Z Berkay Celik, Hidayet Aksu, A Selcuk Uluagac, and Patrick McDaniel. Achieving secure and differentially private computations in multiparty settings. In *2017 IEEE Symp. on Privacy-Aware Computing (PAC)*, pages 49–59, 2017.
- [14] Ahmet Aktay, Shailesh Bavadekar, Gwen Cossoul, John Davis, Damien Desfontaines, Alex Fabrikant, Evgeniy Gabrilovich, Krishna Gadepalli, Bryant Gipson, Miguel Guevara, Chaitanya Kamath, Mansi Kansal, Ali Lange, Chinmoy Mandayam, Andrew Oplinger, Christopher Pluntke, Thomas Roessler, Arran Schlosberg, Tomer Shekel, Swapnil Vispute, Mia Vu, Gregory Wellenius, Brian Williams, and Royce J Wilson. Google covid-19 community mobility reports: Anonymization process description (version 1.1), 2020.
- [15] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. Machine bias. <https://www.propublica.org/article/machine-bias-risk-assessments-in-criminal-sentencing>, 2016.
- [16] Google Apple. Exposure Notification Privacy-preserving Analytics (ENPA).
- [17] Toluwani Aremu and Karthik Nandakumar. PolyKervNets: Activation-free Neural Networks For Efficient Private Inference. In *First IEEE Conference on Secure and Trustworthy Machine Learning*, February 2023.
- [18] Martin Arjovsky. Out of distribution generalization in machine learning. <https://arxiv.org/pdf/2103.02667.pdf>, 2021.
- [19] Giuseppe Ateniese, Luigi V Mancini, Angelo Spognardi, Antonio Villani, Domenico Vitali, and Giovanni Felici. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *International Journal of Security and Networks*, 10:137–150, 2015.
- [20] Eugene Bagdasaryan, Omid Poursaeed, and Vitaly Shmatikov. Differential privacy has disparate impact on model accuracy. In *Advances in Neural Information Processing Systems (NIPS)*, pages 15479–15488, 2019.
- [21] Maria-Florina Balcan, Travis Dick, Yingyu Liang, Wenlong Mou, and Hongyang Zhang. Differentially private clustering in high-dimensional Euclidean spaces. In

- Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 322–331. PMLR, 06–11 Aug 2017.
- [22] Victor Balcer and Salil Vadhan. Differential privacy on finite computers. *Journal of Privacy and Confidentiality*, 9(2), 2019.
 - [23] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In *Advances in Cryptology — CRYPTO 2019*, pages 638–667, 2019.
 - [24] Borja Balle and Yu-Xiang Wang. Improving the Gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 394–403. PMLR, 10–15 Jul 2018.
 - [25] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment*, 12(3):307–320, 2018.
 - [26] Brett K Beaulieu-Jones, William Yuan, Samuel G Finlayson, and Zhiwei Steven Wu. Privacy-preserving distributed deep learning for clinical data. <https://arxiv.org/abs/1812.01484>, 2018.
 - [27] Donald Beaver. Foundations of secure interactive computing. In *Advances in Cryptology — CRYPTO 1991*, pages 377–391, 1991.
 - [28] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC ’88, pages 1–10, New York, NY, USA, January 1988. Association for Computing Machinery.
 - [29] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 1988 ACM Symp. on Theory of Computing*, page 1–10, 1988.
 - [30] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symp. on Operating Systems Principles*, SOSP ’17, pages 441–459, 2017.

- [31] George Robert Blakley. Safeguarding cryptographic keys. In *Int'l Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.
- [32] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. Practical privacy: the SuLQ framework. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '05, pages 128–138, 2005.
- [33] Nicholas Carlini, Steve Chien, Milad Nasr, Shuang Song, Andreas Terzis, and Florian Tramèr. Membership inference attacks from first principles. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1897–1914, 2022.
- [34] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The secret sharer: Evaluating and testing unintended memorization in neural networks. In *USENIX*, 2019.
- [35] Octavian Catrina. Round-Efficient Protocols for Secure Multiparty Fixed-Point Arithmetic. In *2018 International Conference on Communications (COMM)*, pages 431–436, June 2018.
- [36] Octavian Catrina and Sebastiaan de Hoogh. Improved Primitives for Secure Multiparty Integer Computation. In Juan A. Garay and Roberto De Prisco, editors, *Security and Cryptography for Networks*, Lecture Notes in Computer Science, pages 182–199, Berlin, Heidelberg, 2010. Springer.
- [37] Octavian Catrina and Amitabh Saxena. Secure computation with fixed-point numbers. In *Int'l Conf. on Financial Cryptography and Data Security*, pages 35–50, 2010.
- [38] Karan Chadha, John Duchi, and Rohith Kudithipudi. Resampling methods for private statistical inference. *arXiv preprint arXiv:2402.07131*, 2024.
- [39] Alisa Chang and Pritish Kamath. Practical differentially private clustering, 2024.
- [40] Konstantinos Chatzikokolakis, Giovanni Cherubin, Catuscia Palamidessi, and Carmela Troncoso. The bayes security measure. <https://arxiv.org/abs/2011.03396>, 2020.
- [41] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(3), 2011.
- [42] G. Chen, T. Lai, M. K. Reiter, and Y. Zhang. Differentially private access patterns for searchable symmetric encryption. In *2018 IEEE Conf. on Computer Communications (INFOCOM)*, pages 810–818, 2018.

- [43] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In *Advances in Cryptology — EUROCRYPT 2019*, pages 375–403, 2019.
- [44] Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. Lower bounds for differential privacy under continual observation and online threshold queries. In *The Thirty Seventh Annual Conference on Learning Theory*, pages 1200–1222. PMLR, 2024.
- [45] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX Symp. on Networked Systems Design and Implementation (NSDI 17)*, pages 259–282, 2017.
- [46] Paulo Cortez and Alice Silva. Using data mining to predict secondary school student performance. In *European Concurrent Engineering and Future Business Technology Conference*, 2008.
- [47] Christian Covington, Xi He, James Honaker, and Gautam Kamath. Unbiased statistical estimation and valid confidence intervals under differential privacy. *Statistica Sinica*, to appear.
- [48] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT 2000*, pages 316–334, 2000.
- [49] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.
- [50] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, Lecture Notes in Computer Science, pages 285–304, Berlin, Heidelberg, 2006. Springer.
- [51] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Advances in Cryptology – CRYPTO 2012*, pages 643–662, 2012.
- [52] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *Proceedings 2015*

Network and Distributed System Security Symposium, San Diego, CA, 2015. Internet Society.

- [53] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, June 2009.
- [54] Abdulrahman Diaa*, Lucas Fenaux*, Thomas Humphries*, Marian Dietz, Faezeh Ebrahimianghazani, Bailey Kacsmar, Xinda Li, Nils Lukas, Rasoul Akhavan Mahdavi, Simon Oya, Ehsan Amjadian, and Florian Kerschbaum. Fast and private inference of deep neural networks by co-designing activation functions. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2191–2208, 2024.
- [55] Abdulrahman Diaa, Thomas Humphries, and Florian Kerschbaum. {FastLloyd}: Federated, accurate, secure, and tunable {k-Means} clustering with differential privacy. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 2733–2752, 2025.
- [56] Apple Differential Privacy Team. Learning with privacy at scale, December 2017.
- [57] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. *Advances in neural information processing systems*, 30, 2017.
- [58] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, page 3574–3583, 2017.
- [59] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symp.*, page 21, 2004.
- [60] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In Frank Neven, Catriel Beeri, and Tova Milo, editors, *Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA*, pages 202–210. ACM, 2003.
- [61] Jinshuo Dong, Aaron Roth, and Weijie J. Su. Gaussian differential privacy. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 84(1):3–37, 02 2022.
- [62] Jörg Drechsler, Ira Globus-Harris, Audra Mcmillan, Jayshree Sarathy, and Adam Smith. Nonparametric differentially private confidence intervals for the median. *Journal of Survey Statistics and Methodology*, 10(3):804–829, 2022.

- [63] Wenxin Du, Canyon Foot, Monica Moniot, Andrew Bray, and Adam Groce. Differentially private confidence intervals. *arXiv preprint arXiv:2001.02285*, 2020.
- [64] Dheeru Dua and Casey Graff. Uci machine learning repository. <http://archive.ics.uci.edu/ml>, 2021.
- [65] John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. Efficient projections onto the l_1 -ball for learning in high dimensions. In *Proceedings of the 25th International Conference on Machine Learning, ICML '08*, page 272–279, New York, NY, USA, 2008. Association for Computing Machinery.
- [66] Cynthia Dwork. A firm foundation for private data analysis. *Communications of the ACM*, 54(1):86–95, 2011.
- [67] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology – EUROCRYPT 2006*, pages 486–503, 2006.
- [68] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380, 2009.
- [69] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284, 2006.
- [70] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
- [71] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conf.*, pages 316–325, 2014.
- [72] Steven Englehardt. Next steps in privacy-preserving Telemetry with Prio. <https://blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving-telemetry-with-prio/>, 2019. Accessed: 2022-04-08.
- [73] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the 2019 ACM-SIAM Symp. on Discrete Algorithms*, pages 2468–2479, 2019.

- [74] Úlfar Erlingsson, Ilya Mironov, Ananth Raghunathan, and Shuang Song. That which we call private. <https://arxiv.org/abs/1908.03566>, 2019.
- [75] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.
- [76] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conf. on Computer and Communications Security*, pages 1054–1067, 2014.
- [77] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020*, Lecture Notes in Computer Science, pages 823–852, Cham, 2020. Springer International Publishing.
- [78] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In *Advances in Cryptology – CRYPTO 2020*, pages 823–852, 2020.
- [79] Dan Feldman, Amos Fiat, Haim Kaplan, and Kobbi Nissim. Private coresets. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, STOC ’09, pages 361–370. Association for Computing Machinery, 2009.
- [80] Dan Feldman, Chongyuan Xiang, Ruihao Zhu, and Daniela Rus. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pages 3–15. ACM, 2017.
- [81] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets, 2018.
- [82] Karan Ganju, Qi Wang, Wei Yang, Carl A Gunter, and Nikita Borisov. Property inference attacks on fully connected neural networks using permutation invariant representations. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 619–633, 2018.
- [83] Karthik Garimella, Nandan Kumar Jha, and Brandon Reagen. Sisyphus: A Cautionary Tale of Using Low-Degree Polynomial Activations in Privacy-Preserving Deep Learning, November 2021.

- [84] Badih Ghazi, Ravi Kumar, and Pasin Manurangsi. Differentially private clustering: Tight approximation ratios. In *Advances in Neural Information Processing Systems*, volume 33, pages 4040–4054. Curran Associates, Inc., 2020.
- [85] Zahra Ghodsi, Akshaj Kumar Veldanda, Brandon Reagen, and Siddharth Garg. CryptoNAS | Proceedings of the 34th International Conference on Neural Information Processing Systems. *Advances in Neural Information Processing Systems*, 33:16961–16971, 2020.
- [86] Attri Ghosal, Arunima Nandy, Amit Kumar Das, Saptarsi Goswami, and Mrityunjay Panday. A short review on different clustering techniques and their applications. *Emerging Technology in Modelling and Graphics: Proceedings of IEM Graph 2018*, pages 69–83, 2020.
- [87] Arpita Ghosh, Tim Roughgarden, and Mukund Sundararajan. Universally utility-maximizing privacy mechanisms. *SIAM Journal on Computing*, 41(6):1673–1693, 2012.
- [88] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210. PMLR, June 2016.
- [89] Jennifer Gillenwater, Matthew Joseph, Andres Munoz, and Monica Ribero Diaz. A joint exponential mechanism for differentially private top-k. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 7570–7582. PMLR, 2022-07-17/2022-07-23.
- [90] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge university press, 2009.
- [91] Maziar Gomrokchi, Susan Amin, Hossein Aboutalebi, Alexander Wong, and Doina Precup. Where did you learn that from? surprising effectiveness of membership inference attacks against temporally correlated data in deep reinforcement learning. <https://arxiv.org/abs/2109.03975>, 2021.
- [92] Slawomir Goryczka, Li Xiong, and Vaidy Sunderam. Secure multiparty aggregation with differential privacy: A comparative study. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops*, pages 155–163, 2013.

- [93] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *IEEE Symposium on Security and Privacy (S&P) 2019*, 2019.
- [94] Xiaolan Gu, Ming Li, Yueqiang Cheng, Li Xiong, and Yang Cao. PCKV: Locally differentially private correlated key-value data collection with optimized utility. In *29th USENIX Security Symp.*, pages 967–984, 2020.
- [95] Rob Hall, Alessandro Rinaldo, and Larry Wasserman. Differential privacy for functions and functional data. *Journal of Machine Learning Research*, 14(Feb):703–727, 2013.
- [96] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. Principled evaluation of differentially private algorithms using dpbench. In *Proceedings of the 2016 International Conference on Management of Data*, pages 139–154, 2016.
- [97] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [98] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*, pages 1389–1406, 2017.
- [99] Naoise Holohan, Stefano Braghin, Pól Mac Aonghusa, and Killian Levacher. Diffprivlib: the IBM differential privacy library. *ArXiv e-prints*, 1907.02444 [cs.CR], July 2019.
- [100] Hongsheng Hu, Zoran Salcic, Lichao Sun, Gillian Dobbie, and Xuyun Zhang. Source inference attacks in federated learning. <https://arxiv.org/abs/2109.05659>, 2021.
- [101] Zhicong Huang, Wen jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure Two-Party deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, Boston, MA, August 2022. USENIX Association.
- [102] Thomas Humphries*, Rasoul Akhavan Mahdavi*, Shannon Veitch*, and Florian Kerschbaum. Selective mpc: Distributed computation of differentially private key-value statistics. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1459–1472, 2022.

- [103] Thomas Humphries and Florian Kerschbaum. Differentially private simple genetic algorithms. In *Proceedings on Privacy Enhancing Technologies (PoPETs)*, pages 540–558, 2023.
- [104] Thomas Humphries, Tim Li, Shufan Zhang, Karl Knopf, and Xi He. Interpreting the error of differentially private median queries through randomization intervals. *TPDP Workshop 2026*, 2026.
- [105] Thomas Humphries, Zinan Lin, and Sergey Yekhanin. Pe-means: Improved differentially private k-means clustering through private evolution, 2026.
- [106] Thomas Humphries, Simon Oya, Lindsey Tulloch, Matthew Rafuse, Ian Goldberg, Urs Hengartner, and Florian Kerschbaum. Investigating membership inference attacks under data dependencies. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 473–488, 2023.
- [107] Siam Umar Hussain, Mojan Javaheripi, Mohammad Samragh, and Farinaz Koushanfar. COINN: Crypto/ML Codesign for Oblivious Inference via Neural Networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21*, pages 3266–3281, New York, NY, USA, November 2021. Association for Computing Machinery.
- [108] Internet Security Research Group. Let’s Encrypt. <https://letsencrypt.org/>, 2022. Accessed 2022-04-18.
- [109] Matthew Jagielski, Nicholas Carlini, David Berthelot, Alex Kurakin, and Nicolas Papernot. High accuracy and high fidelity extraction of neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, SEC’20, pages 1345–1362, USA, August 2020.
- [110] Matthew Jagielski, Jonathan Ullman, and Alina Oprea. Auditing differentially private machine learning: How private is private sgd? <https://arxiv.org/abs/2006.07709>, 2020.
- [111] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *USENIX Security Symposium*, pages 1895–1912, 2019.
- [112] Bargav Jayaraman, Lingxiao Wang, Katherine Knipmeyer, Quanquan Gu, and David Evans. Revisiting membership inference under realistic assumptions. <https://arxiv.org/abs/2005.10881>, 2020.

- [113] Nandan Kumar Jha, Zahra Ghodsi, Siddharth Garg, and Brandon Reagen. DeepReduce: ReLU Reduction for Fast Private Inference. In *Proceedings of the 38th International Conference on Machine Learning*, pages 4839–4849. PMLR, July 2021.
- [114] Matthew Jones, Huy L. Nguyen, and Thy D Nguyen. Differentially private clustering via maximum coverage. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(13):11555–11563, May 2021.
- [115] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049, 2017.
- [116] Mohammad Mahdi Kamani, Sadegh Farhang, Mehrdad Mahdavi, and James Z. Wang. Targeted data-driven regularization for out-of-distribution generalization. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, page 882–891, New York, NY, USA, 2020. Association for Computing Machinery.
- [117] Marcel Keller. MP-SPDZ: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. <https://eprint.iacr.org/2020/521>.
- [118] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16*, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.
- [119] Daniel Kifer and Ashwin Machanavajjhala. No free lunch in data privacy. In *ACM SIGMOD International Conference on Management of Data (MOD)*, pages 193–204, 2011.
- [120] Daniel Kifer and Ashwin Machanavajjhala. Pufferfish: A framework for mathematical privacy definitions. *ACM Trans. Database Syst.*, 39(1), January 2014.
- [121] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 4961–4973. Curran Associates, Inc., 2021.
- [122] Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207, 1996.

- [123] Alex Krizhevsky et al. Learning multiple layers of features from tiny images. 2009.
- [124] Bogdan Kulynych, Mohammad Yaghini, Giovanni Cherubin, Michael Veale, and Carmela Troncoso. Disparate vulnerability to membership inference attacks. In *Proceedings on Privacy Enhancing Technologies*, pages 460–480, 2022.
- [125] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*, COMPASS '18, New York, NY, USA, 2018. Association for Computing Machinery.
- [126] David Lazar, Yossi Gilad, and Nickolai Zeldovich. Karaoke: Distributed private messaging immune to passive traffic analysis. In *13th USENIX Symp. on Operating Systems Design and Implementation ({OSDI} 18)*, pages 711–725, 2018.
- [127] Yann LeCun. The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>, 1998.
- [128] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise Approximation of Convolutional Neural Networks for Homomorphically Encrypted Data, June 2021.
- [129] Ji Li, Jianghong Wei, Wenfen Liu, Xuexian Hu, and Krzysztof Szczypiorski. Pmdp: A framework for preserving multiparty data privacy in cloud computing. *Sec. and Commun. Netw.*, 2017, 2017.
- [130] Ninghui Li, Min Lyu, Dong Su, and Weining Yang. *Differential privacy: From theory to practice*. Springer, 2017.
- [131] Ninghui Li, Wahbeh Qardaji, Dong Su, Yi Wu, and Weining Yang. Membership privacy: a unifying framework for privacy definitions. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 889–900, 2013.
- [132] Shaohua Li, Kaiping Xue, Bin Zhu, Chenkai Ding, Xindi Gao, David Wei, and Tao Wan. FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8705–8714, 2020.
- [133] Katrina Ligett, Moshe Shenfeld, Tomer Shoham, and Noa Velner-Harris. Differentially private non-parametric confidence intervals. *Journal of Privacy and Confidentiality*, 2025.

- [134] Zinan Lin, Tadas Baltrusaitis, and Sergey Yekhanin. Differentially private synthetic data via APIs 3: Using simulators instead of foundation model. In *ICLR 2025 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2025.
- [135] Zinan Lin, Sivakanth Gopi, Janardhan Kulkarni, Harsha Nori, and Sergey Yekhanin. Differentially private synthetic data via foundation model APIs 1: Images. In *The Twelfth International Conference on Learning Representations*, 2024.
- [136] Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. Dependence makes you vulnerable: Differential privacy under dependent tuples. In *Network and Distributed System Security Symposium (NDSS)*, volume 16, pages 21–24, 2016.
- [137] Jian Liu, Mika Juuti, Yao Lu, and N. Asokan. Oblivious Neural Network Predictions via MiniONN Transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS '17*, pages 619–631, New York, NY, USA, October 2017. Association for Computing Machinery.
- [138] Jiaxiang Liu, Karl Knopf, Yiqing Tan, Bolin Ding, and Xi He. Catch a blowfish alive: a demonstration of policy-aware differential privacy for interactive data exploration. *Proceedings of the VLDB Endowment*, 14(12):2859–2862, 2021.
- [139] S. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.
- [140] Yunhui Long, Lei Wang, Diyu Bu, Vincent Bindschaedler, Xiaofeng Wang, Haixu Tang, Carl A Gunter, and Kai Chen. A pragmatic approach to membership inferences on machine learning models. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 521–534, 2020.
- [141] Donghang Lu, Albert Yu, Aniket Kate, and Hemanta Maji. Polymath: Low-Latency MPC via Secure Polynomial Evaluations and Its Applications. *Proceedings on Privacy Enhancing Technologies*, 2022(1):396–416, January 2022.
- [142] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and its Application to Anonymous Communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS '19*, pages 887–903, New York, NY, USA, November 2019. Association for Computing Machinery.

- [143] Min Lyu, Dong Su, and Ninghui Li. Understanding the sparse vector technique for differential privacy. 10(6):637–648.
- [144] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1):3–es, March 2007.
- [145] Saeed Mahloujifar, Esha Ghosh, and Melissa Chase. Property inference from poisoning. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1569–1569, 2022.
- [146] Rosario Nunzio Mantegna. Fast, accurate algorithm for numerical simulation of lévy stable stochastic processes. *Phys. Rev. E*, 49:4677–4683, May 1994.
- [147] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.
- [148] Sahar Mazloom and S. Dov Gordon. Secure computation with differentially private access patterns. In *Proceedings of the 2018 ACM SIGSAC Conf. on Computer and Communications Security*, pages 490–507, 2018.
- [149] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, FOCS ’07, pages 94–103. IEEE Computer Society.
- [150] Frank D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD ’09, page 19–30, New York, NY, USA, 2009. Association for Computing Machinery.
- [151] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Advances in Cryptology – CRYPTO 2009*, pages 126–142, 2009.
- [152] Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2505–2522, 2020.
- [153] Prashanth Mohan, Abhradeep Thakurta, Elaine Shi, Dawn Song, and David Culler. Gupt: privacy preserving data analysis made easy. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 349–360. ACM, 2012.

- [154] Payman Mohassel and Yupeng Zhang. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38, May 2017.
- [155] Sasi Kumar Murakonda and Reza Shokri. ML privacy meter: Aiding regulatory compliance by quantifying the privacy risks of machine learning. <https://arxiv.org/abs/2007.09339>, 2020.
- [156] Gustav Mårtensson, Daniel Ferreira, Tobias Granberg, Lena Cavallin, Ketil Oppedal, Alessandro Padovani, Irena Rektorova, Laura Bonanni, Matteo Pardini, Milica G Kramberger, John-Paul Taylor, Jakub Hort, Jón Snædal, Jaime Kulisevsky, Frederic Blanc, Angelo Antonini, Patrizia Mecocci, Bruno Vellas, Magda Tsolaki, Iwona Kłoszewska, Hilka Soininen, Simon Lovestone, Andrew Simmons, Dag Aarsland, and Eric Westman. The reliability of a deep learning model in clinical out-of-distribution mri data: A multicohort study. *Medical Image Analysis*, 66:101714, 2020.
- [157] Vaishnavh Nagarajan, Anders Andreassen, and Behnam Neyshabur. Understanding the failure modes of out-of-distribution generalization. <https://arxiv.org/pdf/2010.15775.pdf>, 2021.
- [158] Priyanka Nanayakkara, Johes Bater, Xi He, Jessica Hullman, and Jennie Rogers. Visualizing privacy-utility trade-offs in differentially private data releases. *arXiv preprint arXiv:2201.05964*, 2022.
- [159] Arjun Narayan and Andreas Haeberlen. Djoin: Differentially private join queries over distributed databases. In *10th USENIX Symp. on Operating Systems Design and Implementation (OSDI 12)*, pages 149–162, 2012.
- [160] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP ’08, page 111–125, USA, 2008. IEEE Computer Society.
- [161] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 739–753, 2019.
- [162] Milad Nasr, Shuang Song, Abhradeep Thakurta, Nicolas Papernot, and Nicholas Carlini. Adversary instantiation: Lower bounds for differentially private machine learning. <https://arxiv.org/abs/2101.04535>, 2021.

- [163] Lucien K. L. Ng and Sherman S. M. Chow. {GForce}: {GPU-Friendly} Oblivious and Rapid Neural Network Inference. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2147–2164, 2021.
- [164] Lucien K. L. Ng and Sherman S. M. Chow. SoK: Cryptographic Neural-Network Computation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 497–514, May 2023.
- [165] Thông T. Nguyễn, Xiaokui Xiao, Yin Yang, Siu Cheung Hui, Hyejin Shin, and Junbum Shin. Collecting and analyzing data from smart device users with local differential privacy, 2016.
- [166] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing, STOC '07*, page 75–84, New York, NY, USA, 2007. Association for Computing Machinery.
- [167] Kobbi Nissim and Uri Stemmer. Clustering algorithms for the centralized and local models. In *Proceedings of Algorithmic Learning Theory*, pages 619–653. PMLR, 2018.
- [168] Simon Oya and Florian Kerschbaum. IHOP: Improved statistical query recovery against searchable symmetric encryption through quadratic optimization. In *USENIX Security Symposium*, pages 2407–2424, 2022.
- [169] Liudas Panavas, Amit Sarker, Sara Di Bartolomeo, Ali Sarvghad, Cody Dunne, and Narges Mahyar. Illuminating the landscape of differential privacy: An interview study on the use of visualization in real-world deployments. *IEEE Transactions on Visualization and Computer Graphics*, 2024.
- [170] M Parisot, D Spagnuolo, et al. Property inference attacks on convolutional neural networks: Influence and implications of target model’s complexity. In *18th International Conference on Security and Cryptography, SECRYPT 2021*, pages 715–721, 2021.
- [171] Martin Pettai and Peeter Laud. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st Annual Computer Security Applications Conf.*, pages 421–430, 2015.
- [172] Weiliang Qiu and Harry Joe. *Random Cluster Generation (with Specified Degree of Separation)*, 2023. R package version 1.3.8.

- [173] Deevashwer Rathee, Thomas Schneider, and K. K. Shukla. Improved multiplication triple generation over rings via rlwe-based ahe. In Yi Mu, Robert H. Deng, and Xinyi Huang, editors, *Cryptology and Network Security*, pages 347–359, Cham, 2019. Springer International Publishing.
- [174] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do ImageNet classifiers generalize to ImageNet? In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5389–5400. PMLR, 09–15 Jun 2019.
- [175] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD Int’l Conf. on Management of Data*, pages 603–619, 2020.
- [176] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, Yann Ollivier, and Hervé Jégou. White-box vs black-box: Bayes optimal strategies for membership inference. In *International Conference on Machine Learning*, pages 5558–5567, 2019.
- [177] Shiori Sagawa, Pang Wei Koh, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks for group shifts: On the importance of regularization for worst-case generalization. <https://arxiv.org/pdf/1911.08731.pdf>, 2020.
- [178] Shiori Sagawa, Aditi Raghunathan, Pang Wei Koh, and Percy Liang. An investigation of why overparameterization exacerbates spurious correlations. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8346–8356. PMLR, 13–18 Jul 2020.
- [179] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [180] Elaine Shi, T-H. Hubert Chan, Eleanor Rieffel, Richard Chow, and Dawn Song. Privacy-preserving aggregation of time-series data. In *The 18th Annual Network & Distributed System Security Symp. (NDSS)*, volume 2, pages 1–17, 2011.
- [181] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017.

- [182] Reza Shokri, George Theodorakopoulos, and Carmela Troncoso. Privacy games along location traces: A game-theoretic framework for optimizing location privacy. *ACM Transactions on Privacy and Security (TOPS)*, 19(4):1–31, 2016.
- [183] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, April 2015.
- [184] Liwei Song and Prateek Mittal. Systematic evaluation of privacy risks of machine learning models. In *USENIX Security Symposium*, pages 2615–2632, 2021.
- [185] Uri Stemmer and Haim Kaplan. Differentially private k-means with constant multiplicative error. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.
- [186] Dong Su, Jianneng Cao, Ninghui Li, Elisa Bertino, and Hongxia Jin. Differentially private k-means clustering. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, CODASPY ’16, page 26–37, 2016.
- [187] Dajun Sun, Wei Dong, and Ke Yi. Confidence intervals for private query processing. *Proceedings of the VLDB Endowment*, 17(3):373–385, 2023.
- [188] Lin Sun, Jun Zhao, Xiaojun Ye, Shuo Feng, Teng Wang, and Tao Bai. Conditional analysis for key-value data with local differential privacy, 2019.
- [189] Kunal Talwar, Shan Wang, Audra McMillan, Vitaly Feldman, Pansy Bansal, Bailey Basile, Aine Cahill, Yi Sheng Chan, Mike Chatzidakis, Junye Chen, Oliver R. A. Chick, Mona Chitnis, Suman Ganta, Yusuf Goren, Filip Granqvist, Kristine Guo, Frederic Jacobs, Omid Javidbakht, Albert Liu, Richard Low, Dan Mascenik, Steve Myers, David Park, Wonhee Park, Gianni Parsa, Tommy Pauly, Christian Priebe, Rehan Rishi, Guy N. Rothblum, Congzheng Song, Linmao Song, Karl Tarbe, Sebastian Vogt, Shundong Zhou, Vojta Jina, Michael Scaria, and Luke Winstrom. Samplable anonymous aggregation for private federated data analysis. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS ’24, page 2859–2873, New York, NY, USA, 2024. Association for Computing Machinery.
- [190] Sijun Tan, Brian Knott, Yuan Tian, and David J. Wu. CryptGPU: Fast Privacy-Preserving Machine Learning on the GPU. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1021–1038, May 2021.

- [191] Raphael R Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ -private information retrieval. *Proceedings on Privacy Enhancing Technologies*, 2016(4):184–201, 2016.
- [192] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction apis. In *25th USENIX Security Symposium (USENIX Security 16)*, volume 16, pages 601–618, 2016.
- [193] Toan Tran, Arturs Backurs, Zinan Lin, Victor Reis, Li Xiong, and Sergey Yekhanin. Differentially private synthetic data via apis 4: Tabular data. *arXiv preprint arXiv:2606.08259*, 2026.
- [194] Michael Carl Tschantz, Shayak Sen, and Anupam Datta. SoK: Differential Privacy as a Causal Property. In *IEEE Symposium on Security and Privacy (SP)*, pages 354–371, 2020.
- [195] Nirvan Tyagi, Yossi Gilad, Derek Leung, Matei Zaharia, and Nikolai Zeldovich. Stadium: A distributed metadata-private messaging system. In *Proceedings of the 26th Symp. on Operating Systems Principles*, page 423–440, 2017.
- [196] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nikolai Zeldovich. Vuvuzela: Scalable private messaging resistant to traffic analysis. In *Proceedings of the 25th Symp. on Operating Systems Principles*, page 137–152, 2015.
- [197] Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. Dp-cryptography: marrying differential privacy and cryptography in emerging applications. *Communications of the ACM*, 64(2):84–93, 2021.
- [198] Lauren Watson, Chuan Guo, Graham Cormode, and Alex Sablayrolles. On the importance of difficulty calibration in membership inference attacks. <https://arxiv.org/abs/2111.08440>, 2021.
- [199] Genqiang Wu, Yeping He, Jingzheng Wu, and Xian Yao Xia. Inherit differential privacy in distributed setting: Multiparty randomized function computation. In *2016 IEEE Trustcom/BigDataSE/ISPA*, pages 921–928, 2016.
- [200] Siyuan Xia, Beizhen Chang, Karl Knopf, Yihan He, Yuchao Tao, and Xi He. Dpgraph: A benchmark platform for differentially private graph analysis. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2808–2812, 2021.

- [201] Chulin Xie, Zinan Lin, Arturs Backurs, Sivakanth Gopi, Da Yu, Huseyin A Inan, Harsha Nori, Haotian Jiang, Huishuai Zhang, Yin Tat Lee, Bo Li, and Sergey Yekhanin. Differentially private synthetic data via foundation model APIs 2: Text. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 54531–54560. PMLR, 21–27 Jul 2024.
- [202] Zuobin Xiong, Zhipeng Cai, Daniel Takabi, and Wei Li. Privacy threat and defense for federated learning with non-i.i.d. data in aiots. *IEEE Transactions on Industrial Informatics*, 18(2):1310–1321, 2022.
- [203] Jiayuan Ye, Aadyaa Maddi, Sasi Kumar Murakonda, Vincent Bindschaedler, and Reza Shokri. Enhanced membership inference attacks against machine learning models. <https://arxiv.org/abs/2111.09679>, 2021.
- [204] Q. Ye, H. Hu, X. Meng, and H. Zheng. PrivKV: Key-value data collection with local differential privacy. In *2019 IEEE Symp. on Security and Privacy (SP)*, pages 317–331, 2019.
- [205] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. Privacy risk in machine learning: Analyzing the connection to overfitting. In *IEEE Computer Security Foundations Symposium (CSF)*, pages 268–282, 2018.
- [206] Lei Yu, Ling Liu, Calton Pu, Mehmet Emre Gursoy, and Stacey Truex. Differentially private model publishing for deep learning. In *IEEE Symposium on Security and Privacy (SP)*, pages 332–349, 2019.
- [207] Jun Zhang, Xiaokui Xiao, Yin Yang, Zhenjie Zhang, and Marianne Winslett. Privgene: Differentially private model fitting using genetic algorithms. New York, NY, USA, 2013. Association for Computing Machinery.
- [208] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: A new data clustering algorithm and its applications. *Data Mining and Knowledge Discovery*, 1(2):141–182, 1997.
- [209] Wanrong Zhang, Shruti Tople, and Olga Ohrimenko. Leakage of dataset properties in {Multi-Party} machine learning. In *USENIX Security Symposium*, pages 2687–2704, 2021.

Appendices

Appendix A

Privacy of the One-Sided Dummy Solution

For a description of the one-sided dummy solution, see Section 3.4. We consider an implementation of our one-sided dummy solution using the strategy by Mazloom et al. [148]. This strategy uses a modified version of the Two-sided Geometric distribution (an over-approximation) to add noise. For completeness, we provide a version of their proofs, in our context, using the standard two-sided Geometric distribution.

A.1 Background on the Geometric Mechanism

First, we recall that the PMF of the two-sided geometric distribution is

$$\mathbb{G}_2(z; \alpha) = \frac{1 - \alpha}{1 + \alpha} \alpha^{|z|} \quad (\text{A.1})$$

for $\alpha \in [0, 1]$.

Definition A.1.1 (Geometric Mechanism). *Let $f : \mathcal{X}^n \mapsto \mathbb{R}^k$. The Geometric Mechanism is defined as*

$$G(X) = f(X) + \mathbb{G}_2(e^{-\epsilon/\Delta})^n, \quad (\text{A.2})$$

where $\mathbb{G}_2(e^{-\epsilon/\Delta})^n$ represents n i.i.d draws from the Geometric distribution with parameter $\alpha = e^{-\epsilon/\Delta}$

Theorem A.1.1 ([87]). The Geometric Mechanism satisfies ϵ -DP

Proof Sketch

$$\frac{\mathbb{G}_2(z; \alpha)}{\mathbb{G}_2(z + \Delta; \alpha)} = \frac{\frac{1-\alpha}{1+\alpha} \alpha^{|z|}}{\frac{1-\alpha}{1+\alpha} \alpha^{|z+\Delta|}} \leq \alpha^{-|\Delta|} = e^\epsilon \quad (\text{A.3})$$

where the inequality comes from triangle inequality and the last equality from setting $\alpha = e^{-\epsilon/\Delta}$.

A.2 Privacy proof

We consider the one-sided dummy mechanism, defined as

$$D_k(\mathcal{S}) = q_k(\mathcal{S}) + \max(0, \beta + \mathbb{G}_2(e^{-\epsilon})) \quad (\text{A.4})$$

where β is a constant chosen such that $\Pr[\beta + \mathbb{G}_2(e^{-\epsilon}) < 0]$ is negligible.

Theorem A.2.1 ([148]). $D_k(\mathcal{S})$ is (ϵ, δ) -DP

Proof. Using Theorem A.1.1 we can obtain the approximate DP guarantee as follows. First, we define a set $\mathcal{B}$ of all outcomes where we add negative noise. Conditioning on not being in this set we get

$$\Pr[D(\mathcal{S}) \in Z \setminus \mathcal{B}] = \prod_{k=0}^n \Pr[D_k(\mathcal{S}) \in Z_k \setminus \mathcal{B}] \quad (\text{A.5})$$

$$\leq \prod_{k=0}^n e^\epsilon \Pr[D_k(\mathcal{S}') \in Z_k \setminus \mathcal{B}] \quad (\text{A.6})$$

$$= e^\epsilon \Pr[D(\mathcal{S}') \in Z \setminus \mathcal{B}] \quad (\text{A.7})$$

Now, using this, we obtain the final result

$$\Pr[D(\mathcal{S}) \in Z] = \Pr[D(\mathcal{S}) \in Z \setminus \mathcal{B}] + \Pr[D(\mathcal{S}) \in \mathcal{B}] \quad (\text{A.8})$$

$$\leq e^\epsilon \Pr[D(\mathcal{S}') \in Z \setminus \mathcal{B}] + \Pr[D(\mathcal{S}) \in \mathcal{B}] \quad (\text{A.9})$$

$$\leq e^\epsilon \Pr[D(\mathcal{S}') \in Z] + \Pr[D(\mathcal{S}) \in \mathcal{B}] \quad (\text{A.10})$$

Thus as long as we choose β such that $\Pr[D(\mathcal{S}) \in \mathcal{B}] \leq \delta$, we have approximate DP. $\square$

A.3 How many dummies to add

Next, we specify how to choose β to satisfy approximate DP. To do this, we begin by calculating

$$Pr[\beta + \mathbb{G}_2(\alpha) < 0] = \sum_{z=-\infty}^0 \frac{1-\alpha}{1+\alpha} \alpha^{|z-\beta|} \quad (\text{A.11})$$

$$= \sum_{z=2\beta}^{\infty} \frac{1-\alpha}{1+\alpha} \alpha^{|z-\beta|} \quad (\text{A.12})$$

$$= \frac{1-\alpha}{1+\alpha} \alpha^{\beta} \sum_{z=0}^{\infty} \alpha^z \quad (\text{A.13})$$

$$= \frac{\alpha^{\beta}}{1+\alpha} \quad (\text{A.14})$$

then, applying the union bound we get that

$$Pr[D(\mathcal{S}) \in \mathcal{B}] \leq \frac{\lambda \alpha^{\beta}}{1+\alpha} \leq \delta. \quad (\text{A.15})$$

Finally, we can rearrange (A.15) to obtain a formula for β

$$\beta > \log_{\alpha} \left(\frac{\delta(1+\alpha)}{\lambda} \right) = -\frac{1}{\epsilon} \ln \left(\frac{\delta(1+e^{-\epsilon})}{\lambda} \right). \quad (\text{A.16})$$

A.4 Comparison with our protocol

The primary difference between the one-sided dummy solution and our final protocol is how the number of dummies is selected. In the one-sided dummy solution, dummies are chosen from a geometric distribution shifted by some amount, which we call β . The expected value of a single sample of the shifted and truncated two-sized geometric distribution is at least β . Additionally, each element is given to all nodes rather than a subset of them. Considering these changes, the client-to-server communication of the one-sided dummy solution is $\Omega(|\mathcal{S}|\ell + n\beta\ell)$. The computation cost of this protocol in the MPC phase is the same as our solution.

Figure A.1 demonstrates the difference in the expected number of dummies for the one-sided dummy solution and our final solution (selective MPC). It is clear that selective MPC produces less dummies and is more efficient.

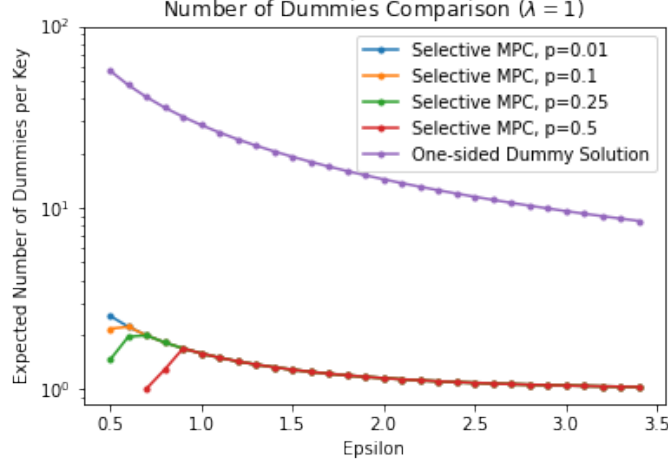


Figure A.1: We consider the expected number of dummies for various epsilons. r is chosen to minimize the epsilon spent and where applicable delta is fixed to $\delta = 2^{-40}$.

A.5 Using the noise from dummies

A natural question is if the dummies used to achieve a DP leakage could also be used to make the output private (instead of adding additional Laplace noise in a distributed manner). That is, instead of discarding the dummies, we may count them as real values to add noise to the frequency statistic. We remark that this is not possible in selective MPC. Due to the fact that we add dummies following a one-sided geometric distribution, the output cannot satisfy DP (even approximate DP as the δ is prohibitively large). However, the dummies could be used in the one-sided dummy solution. Specifically, for frequency estimation, we have already paid the privacy cost in ϵ_L , and could simply publish the view of the computation node as the noisy frequency. One would likely want to subtract β from the final result to improve accuracy. This is private since β is derived from public values ϵ , δ , and λ . The downside to this approach is that it inherits the downsides of the one-sided dummy solution: approximate DP instead of pure DP and increased communication due to a large number of dummies. It is also not clear how this approach could be extended to mean estimations.