Secure Noise Sampling for Differentially Private Collaborative Learning

Olive Franzese[1]*, **Congyu Fang**[1], **Radhika Garg**[2],
Somesh Jha[3,4], **Nicolas Papernot**[1], **Xiao Wang**[2], **Adam Dziedzic**[5]*
[1]University of Toronto and Vector Institute, [2]Northwestern University,
[3]University of Wisconsin-Madison, [4]Google,
[5]CISPA Helmholtz Center for Information Security

June 2, 2025

Abstract

Differentially private stochastic gradient descent (DP-SGD) trains machine learning (ML) models with formal privacy guarantees for the training set by adding random noise to gradient updates. In collaborative learning (CL), where multiple parties jointly train a model, noise addition occurs either (i) before or (ii) during secure gradient aggregation. The first option is deployed in distributed DP methods, which require greater amounts of total noise to achieve security, resulting in degraded model utility. The second approach preserves model utility but requires a secure multiparty computation (MPC) protocol. Existing methods for MPC noise generation require tens to hundreds of seconds of runtime per noise sample because of the number of parties involved. This makes them impractical for collaborative learning, which often requires thousands or more samples of noise in each training step.

We present a novel protocol for MPC noise sampling tailored to the collaborative learning setting. It works by constructing an approximation of the distribution of interest which can be efficiently sampled by a series of table lookups. Our method achieves significant runtime improvements and requires much less communication compared to previous work, especially at higher numbers of parties. It is also highly flexible – while previous MPC sampling methods tend to be optimized for specific distributions, we prove that our method can generically sample noise from statistically close approximations of *arbitrary* discrete distributions. This makes it compatible with a wide variety of DP mechanisms. Our experiments demonstrate the efficiency and utility of our method applied to a discrete Gaussian mechanism for differentially private collaborative learning. For 16 parties, we achieve a runtime of 0.06 seconds and 11.59 MB total communication per sample, a 230× runtime improvement and 3× less communication compared to the prior state-of-the-art [34] for sampling from discrete Gaussian distribution in MPC.

1 Introduction

Differentially private stochastic gradient descent (DP-SGD) [1] is a model training algorithm which adds calibrated amounts of random noise to gradient updates in order to obtain rigorous differential privacy (DP) guarantees [8] for the training data. In collaborative learning (CL) algorithms [21, 25, 14, 20], which allow many parties to work together to train a model, there is no central party who is trusted to add the noise. Adapting DP-SGD to this setting requires the addition of noise either (i) before or (ii) during secure aggregation of gradient updates.

*Correspondance to: olive.franzese@gmail.com and adam.dziedzic@cispa.de

Table 1: **Comparison with prior work [34].** Comparison of the time and total communication required per sample from the discrete Gaussian distribution with a standard deviation of 967, as used in our DP-CL experiments. Prior work is evaluated using GMW protocol with triple generation using OT-based (`semi2k`) and HE-based (`temi`) protocols.

Parties		Ours	Prior (OT)	Prior (HE)
2	LAN (s)	0.09	0.27	1.24
	WAN (s)	0.1	598	631
	Comm. (MB)	0.72	0.29	0.29
4	LAN (s)	0.07	15.05	3.36
	WAN (s)	0.08	1.457K	1.459K
	Comm. (MB)	1.91	6.82	1.67
8	LAN (s)	0.06	27.49	6.57
	WAN (s)	0.08	2.612K	2.613K
	Comm. (MB)	3.3	31.51	7.71
16	LAN (s)	0.06	50.17	12.78
	WAN (s)	0.25	4.942K	4.916K
	Comm. (MB)	11.59	134.36	32.91

The former option, utilized in distributed DP [14, 10, 19, 20] methods, is relatively straightforward: each party adds some noise locally to their update. This confers the aggregated output with a DP guarantee due to the combined noise. However, this approach introduces a vulnerability. If multiple parties collude by sharing their local noise values, they can subtract them from the aggregated output to partially de-noise it. To compensate for this attack each party must add a larger amount of noise, proportional to the number of expected colluding adversaries. This extra noise results in degraded model utility compared to the centralized setting.

Alternatively, DP noise may be sampled inside of a secure multiparty computation (MPC) protocol for gradient aggregation. MPC allows a set of parties to compute a function on a pool of private inputs, while guaranteeing that each party's input is confidential [5, 22, 34]. Using MPC to sample noise means that no party views any part of the noise until after it has been combined with the aggregated updates. As a result, this approach achieves maximal utility in collaborative learning without requiring a trusted central party. However, it comes at the cost of increased computational overhead.

Existing methods for MPC noise generation are either limited to two-parties [5] or do not scale well to higher numbers of parties [22, 34]. Runtimes in the tens of seconds per sample with as few as four parties (Table 1) pose steep limitations on their practicality in the collaborative learning setting, which may involve tens to hundreds of parties and thousands or more noise samples per training round.

In this work we present a method for MPC noise sampling with greatly improved scaling to higher number of parties, making it more suitable for collaborative learning. In addition, unlike previous MPC noise generation methods which tend to be optimized for particular distributions [5, 22, 34], our method is highly generic. It can accommodate any discrete distribution with a known probability mass function, making it versatile to different applications. For example, DP algorithms are an ongoing area of study. Noise from a variety of different distributions may be utilized in existing and future work [12].

These advances are made possible by our novel approach to the problem. We present the overview of our method in Figure 1. Rather than applying generic MPC to compute the sampling algorithm

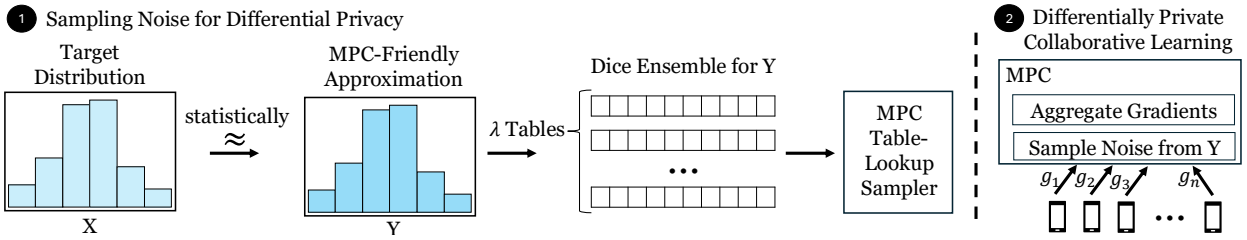

Figure 1: **Overview of the Method.** ❶ Sampling Noise for Differential Privacy. Our method takes as input a target distribution X generically. It finds Y, an approximation of X which is statistically indistinguishable, and can be sampled using a series of random table lookups which we call a "dice ensemble". Algorithm 4 constructs a dice ensemble appropriate to the target distribution. Then, Protocol 4 can be used to sample Y efficiently in MPC. Our method substantially enhances scalability to higher numbers of parties while supporting any discrete distribution. ❷ Differentially Private Collaborative Learning. Our method is leveraged with a secure gradient aggregation protocol to provide a collaborative learning protocol with differential privacy guarantees.

for a distribution of interest X as in previous work [5, 22, 34], our method takes X generically as input and constructs Y, a statistically close approximation which can be sampled as a series of table lookups. We then apply an efficient private table lookup protocol to realize an MPC sampler for Y, resulting in much more scalable noise generation.

Summary of Contributions. We propose a new method for sampling random noise in MPC with the following contributions:

- *Efficiency.* Our method achieves substantial improvements in runtime and communication compared to previous work, especially at higher numbers of parties. For 32 parties, our method samples from the discrete Gaussian distribution in 0.21 seconds with 42 MB of total communication, representing 450× and 13× improvement over previous work respectively (see Table 1). We also note that our protocol achieves significant improvement even for 4 parties.
- *Flexibility.* Our method is highly generic. It takes the probability mass function of any discrete distribution as input, and compiles it into an MPC sampler. We prove that the outputs of our sampler are statistically indistinguishable from the input distribution.
- *Collaborative Learning Benchmarks.* We adapt previous work to derive parameter settings appropriate to differentially private collaborative learning, and benchmark them in terms of efficiency and model utility. Our experiments show that the reduced noise afforded via MPC sampling results in models with improved accuracy compared to distributed DP that needs to account for colluding clients.
- *Code.* We provide our code at `https://github.com/cleverhans-lab/Secure_Noise_Sampling_DP_CL`.

2 Background

2.1 Noise Sampling for Differentially Private Collaborative Learning

The two prior approaches for sampling noise in collaborative learning that are most relevant to our work are:

Distributed DP . In this approach [14, 10, 19, 20], parties locally sample noise and use an MPC protocol for aggregation such that the aggregated noise follows a certain distribution which confers a DP guarantee. For example, in [20], the authors demonstrate that if each party locally samples a discrete Gaussian noise independently, the summation of those discrete Gaussian samples forms a distribution X; they further prove the DP guarantee for the algorithm that adds noise sampled from X. However, for security against t colluding (out of n parties), each party needs to add noises as if only $n - t$ parties will sample noise to meet the desired aggregate distribution X. Thus, for high corruption thresholds, these works add a large amount of noise to the data, reducing the model's utility.

DP using MPC . In this setting [5, 22, 34], parties use an MPC protocol to sample from a desired distribution required for the DP guarantee. These works focus on designing efficient algorithms or smaller circuits for sampling in MPC from a specific distribution like the Gaussian or Laplace distributions.

Dwork et al. [10] observe that the generation of a geometric sample can be reduced to independent Bernoulli samples for each bit with a different bias. For a κ bit Geometric sample with probability p, the i-th bit b_i is defined by Bernoulli sample with probability $p^{2^i}/(1+p)^{2^i}$. [5] use this observation with their efficient sampling algorithm for biased coins to generate Geometric noise for DP in 2-PC. They design an efficient oblivious stack algorithm with `pop` and `reset` operations in order to hide the access pattern and avoid iterating for the binary expansion of the bias.

In [4], the authors propose an algorithm for sampling from discrete Gaussian distribution for differential privacy using Bernoulli, Geometric, and Laplace distribution for central DP applications. [34] follows a similar structure to the central DP algorithm for sampling from discrete Gaussian distribution in MPC. They use the method described in [10] to generate geometric samples from Bernoulli samples. Their main contribution is to improve the sampling of Bernoulli noise when $p = e^{-\gamma}$ and γ is private. They observe that this is only required to go from Laplace noise to Gaussian noise in [4], where $\gamma = a/b$ and b is public while a is private and has 2κ bits. Thus, they can sample 2κ biased coins with public biases and combine them according to the bit decomposition of a. Additionally, they also optimize the acceptance rate when going from the Laplace sample to the Gaussian sample. Further, [16] improves [34] by updating their Bernoulli sampling algorithm to use the oblivious stack algorithm from [5].

2.2 Differentially Private Collaborative Learning Algorithms

In a collaborative training setting, two types of granularity for privacy are often considered based on the nature of the clients: (i) client-level privacy [25, 18, 20], useful for clients like personal devices (e.g., cellphone) that contains information about one individual; and (ii) data point-level privacy [21, 14], useful for clients that hold data from many different individuals (e.g., hospitals) where each individual's privacy shall be considered.

Federated Learning (FL) [24] is one of the earliest proposed collaborative training methods. In FL, each client uses local data to obtain a model update and shares it with the server for aggregation. In this work, we focus on leveraging the proposed method of noise sampling in MPC on FL to protect data point-level privacy.

Other than FL-based collaborative training methods, another type of DP-CL method involves querying teacher models trained on local private data to label public data points (with DP). The labeled pairs are then used to train a centralized student model. Such frameworks include Private Aggregation of Teach Ensembles (PATE) [28] and Confidential and Private Collaborative (CaPC) Learning [6]. Our noise sampling in MPC can also be used to obtain the DP guarantees in these frameworks, for example, by replacing the role of Privacy Guardian (a third-party responsible for

noise addition) in CaPC.

2.3 DP Fine-Tuning for Soft Prompts

Soft prompts refer to the list of weights prepended to the embeddings of the inputs to language models (LM). These weights can be learned in the same manner as regular model weights on downstream tasks, and they can leak private information just as the model weights can. Soft prompts have the benefit that they are usually low in dimension so the cost of training is also low. Prior work [7] has demonstrated DP fine-tuning for soft prompts in a centralized manner. In this work, we extend this approach to a collaborative training scenario.

3 Preliminaries & Notation

3.1 Secure Multiparty Computation

Secure multi-party computation (MPC) enables a group of parties to collaboratively compute a function f on their private data while revealing nothing beyond the output. The function f is represented as a circuit C with boolean and/or arithmetic gates that can be evaluated using a generic MPC protocol. MPC protocols can be designed to achieve different levels of security depending on the adversarial model. In this work, we focus on the *semi-honest* security model with *all-but-one* corruption. In the semi-honest model, the corrupt parties may collude to infer additional information without deviating from the protocol. In this work, we use the Boolean to Arithmetic shares protocol from [17] to realize the functionality for converting Boolean shares of random indices to their bitwise encryptions. We also use a threshold homomorphic encryption (THE) scheme to enable the parties to do a majority of the computation non-interactively/locally.

3.2 Representing Approximation Error in Constructed Distributions

Given a discrete, finite sample space Ω, random variable R which maps Ω to $\mathbb{R}$, and a probability mass function $f : \mathbb{R} \mapsto [0, 1]$, in this work we are interested in constructing approximations of f. We will often construct an augmented sample space $\Omega \cup \{\perp\}$ where $\perp \notin \Omega$ is a special element which represents approximation error. Since Ω is discrete and finite, we can always construct $R' : \Omega \cup \{\perp\} \mapsto \mathbb{R}$ such that $R'(\perp) = \theta$ where $\theta \neq R(x)$ for all $x \in \Omega$. However, for brevity in the remainder of the paper we will abuse notation by rendering some probability mass functions with the signature $f : \mathbb{R} \cup \{\perp\} \mapsto [0, 1]$. This gives us $f(\perp)$ as convenient shorthand for discussing the amount of probability mass allocated to this special error element.

3.3 Statistical Indistinguishability

We use a formalization of statistical indistinguishability from [15]. Given two sequences of discrete distributions parameterized by a statistical security parameter λ, $X = (X_\lambda)_{\lambda\in\mathbb{N}}$ and $Y = (Y_\lambda)_{\lambda\in\mathbb{N}}$, we will use the following formulation of total variation distance, also parameterized by λ:

$$SD_{X,Y}(\lambda) := \frac{1}{2} \cdot \sum_{z\in 0,1^*} \left| \Pr\left[X(1^\lambda) = z\right] - \Pr\left[Y(1^\lambda) = z\right] \right|.$$

We say that X and Y are *statistically indistinguishable* if $SD_{X,Y}(\lambda)$ is a negligible function of λ. That is, if for any polynomial $\mathtt{p} : \mathbb{N} \to \mathbb{R}^+$ there exists an integer N such that for all $\lambda \geq N$ we have

$$SD_{X,Y}(\lambda) \leq \frac{1}{\mathtt{p}(\lambda)}.$$

3.4 Discrete Gaussian Distribution

Let scale and location parameter $\sigma, \mu > 0$. The discrete Gaussian [4] is a probability distribution supported on the integers $\mathbb{Z}$ denoted by $\mathcal{N}_{\mathbb{Z}}(\mu, \sigma^2)$ and defined as the follows:

$$\forall x \in \mathbb{Z}, \qquad \mathop{\mathbb{P}}_{X \leftarrow \mathcal{N}_{\mathbb{Z}}(\mu,\sigma^2)}[X = x] = \frac{\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)}{\sum_{y \in \mathbb{Z}} \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right)}.$$

3.5 Differential Privacy

To reason about and bound privacy leakage in machine learning training algorithms, differential privacy (DP) [8, 9, 11, 12] is the current gold standard and working definition of privacy:

(ϵ, δ)-DP: A randomized algorithm $M : \mathcal{X} \mapsto \mathcal{Y}$ satisfies (ϵ, δ)-DP if for any adjacent datasets that differ by only one data record $x, x' \in \mathcal{X}$, some algorithm output $S \subset \mathcal{Y}$ the algorithm M satisfies

$$\Pr[M(x) \in S] \leq e^{\epsilon} \Pr\left[M\left(x'\right) \in S\right] + \delta.$$

The ϵ is also known as the privacy budget, which quantifies the privacy leakage of the algorithm M. When $\delta = 0$, it is pure differential privacy (ϵ-DP). When $\delta > 0$, it is approximated DP and it is a common relaxation of pure DP.

Concentrated DP (CDP) [13, 3]: A randomized $M : \mathcal{X} \mapsto \mathcal{Y}$ satisfies $\frac{1}{2}\epsilon^2$-DP iff for any adjacent datasets that differ by only the addition or removal of one data record $x, x' \in \mathcal{X}$, $\mathcal{D}_\alpha(M(x)||M(x')) \leq \frac{1}{2}\epsilon^2\alpha, \quad \forall \alpha \in (1, \infty)$, where $\mathcal{D}_\alpha(\cdot||\cdot)$ is the Rényi divergence of order α, defined as $\mathcal{D}_\alpha(P||Q) = \frac{1}{\alpha-1}\log \mathop{\mathbb{E}}_{X \leftarrow P}\left(\frac{P(X)}{Q(X)}\right)^{\alpha-1}$.

Theorem 1 (Privacy for Multivariate Discrete Gaussian [4])**.** *Let $\sigma > 0$ and $\varepsilon > 0$. Let $q\colon \mathcal{X} \to \mathbb{Z}^d$ satisfy $\Delta^2/\sigma^2 \leq \varepsilon^2$ for all $x, x' \in \mathcal{X}$ differing on a single entry, $\Delta = ||q(x) - q(x')||_2$. Define a randomized algorithm $M\colon \mathcal{X}^n \to \mathbb{Z}^d$ by $M(x) = q(x) + Y$ where $Y_j \leftarrow \mathcal{N}_{\mathbb{Z}}(0, \sigma^2)$ independently for all $j \in [d]$. Then M satisfies $\frac{1}{2}\varepsilon^2$-concentrated differential privacy.*

4 Method

We present a secure multiparty computation protocol which allows a set of parties to efficiently sample a hidden noise value from a distribution. For many distributions of interest in differential privacy, sampling typically requires the computation of non-linear functions which are inefficient in MPC. The key idea of our method is to construct statistically indistinguishable approximations of target distributions which can be sampled via more efficient means. In particular, we design a method for approximating distributions by a series of uniform random table lookups. We present the overview of our method in Figure 1.

Section 4.1 shows how we can compose uniform table lookups to construct a statistically indistinguishable approximation of any discrete probability distribution. Section 4.2 shows an efficient multiparty computation protocol for sampling from these approximated distributions. Section 4.3 discusses the application of our MPC noise sampler to differentially private collaborative learning by way of the discrete Gaussian mechanism.

4.1 Approximating Distributions with Table Lookups

Section Outline. In a uniform table lookup, we fix a table of values and output an entry of the table uniform randomly (Definition 1). This operation is equivalent to rolling a fair die, and can be computed highly efficiently within an MPC protocol as we will see in Section 4.2. In this section, we

show how we can approximate a sampling process for *any* discrete distribution of interest by using a sequence of uniform random table lookups, and characterize a tradeoff between statistical distance and number of tables.

To achieve this result, we begin with Algorithm 1 which approximates a target distribution using a random single table lookup. Lemma 3 proves a relationship between approximation error and the size of the table. With this as a building block, we then present Algorithm 4 which achieves a greatly improved tradeoff via the composition of lookups from multiple smaller tables. Theorem 6 shows that the statistical distance between the target and approximated distributions declines *exponentially* as a function of the number of tables. We leverage this result to show statistical indistinguishability.

4.1.1 Approximation with One Table Lookup

While we use the term "table lookup" to evoke mechanical similarities between our method and other cryptographic protocols, tables are highly general objects. While proving our theoretical results, we will disambiguate using the term "die" which specifically indicates a probability distribution induced by uniformly sampling entries from a table (Definition 1).

Definition 1. *A* **die** d *is an n-sized array that encodes a finite discrete probability distribution. To sample from the distribution that d encodes, we take $i \xleftarrow{\$} [n]$, a uniform random sample over the indices in the array, and output $d[i]$, the i^{th} element in the array. Thus the probability mass function f_d corresponding to d is precisely*

$$f_d(x) \equiv \frac{\mathtt{freq}_d(x)}{n}$$

where $\mathtt{freq}_d(x)$ is a function that gives the number of times the element x appears in d.

We can construct a die to approximate any given discrete probability distribution. For example, a sampling algorithm defined over 32-bit seeds can be perfectly encoded by a die with 2^{32} faces. Algorithm 1 formalizes a simple method for approximating distributions using single dice. We will analyze it briefly and then iterate on this method to acquire finer approximations.

Algorithm 1 1DA (1-Die Approximation)

Input: A probability mass function $f : \mathbb{R} \mapsto [0, 1]$ corresponding to a finite discrete probability distribution; a die size n.

Output: A die d that approximates f.

1: $d \leftarrow$ an n-sized array, all elements initialized to $\perp$.
2: **for all** $x \in supp(f)$ **do**
3: $\quad t \leftarrow \lfloor f(x) \cdot n \rfloor$
4: $\quad$ find t entries of d that contain $\perp$. Write x in these entries instead.
5: **end for**
6: return d.

Lemma 1. *$d = 1DA(f, n)$ is defined for arbitrary $n > 0 \in \mathbb{N}$, and arbitrary probability mass functions f with finite support.*

Proof: To see this, observe that step 4 of Algorithm 1 will never fail to find t entries that contain $\perp$. This is because by the definition of a discrete probability distribution, $\sum_{x \in supp(f)} f(x) = 1$ and thus $\sum_{x \in supp(f)} \lfloor f(x) \cdot n \rfloor \leq n$. The left hand side of the inequality describes the total number of $\perp$ entries which must be overwritten, and n is the number of $\perp$ entries when d is initialized. Since we assume that f is supported on a finite set, we also know that the for loop will always terminate. $\square$

We measure the effectiveness of f_d as an approximation for f by comparing the respective probability masses allocated to elements of $supp(f)$.

Definition 2. *Given a target probability mass function $f : \mathbb{R} \mapsto [0,1]$ and an approximation probability mass function $g : \mathbb{R} \cup \{\perp\} \mapsto [0,1]$, the* **approximation error** *function $\zeta : \mathbb{R} \mapsto \mathbb{R}$ is defined*

$$\zeta(x) \equiv |f(x) - g(x)|.$$

When it is clear from context that the function g is constructed to approximate the function f, we will also broaden this notation so that $\zeta(g)$ indicates the total additive error over the support of f. Explicitly,

$$\zeta(g) \equiv \sum_{x \in supp(f)} |f(x) - g(x)|.$$

While we define the error function ζ using an absolute value for intuition and compatibility with statistical distance, we note that Algorithm 1 is constructed such that $f(x) \geq f_d(x) \forall x \in supp(f)$, and both f and f_d are always non-negative. Accordingly, we can 'drop' the absolute value in many contexts. This is formalized in Fact 1.

Fact 1. *For arbitrary $d = 1DA(f,n)$ we have $0 \leq f_d(x) < f(x)$ for any $x \in supp(f)$ since*

$$f_d(x) \equiv \frac{\lfloor f(x) \cdot n \rfloor}{n} \leq f(x)$$

and $f(x)$ is always non-negative by the definition of a probability mass function. Consequently, we can drop the absolute value when considering approximations made using Algorithm 1. Thus we have

$$\zeta(f_d) = \sum_{x \in supp(f)} f(x) - f_d(x).$$

We use the special element $\perp$ to represent approximation error in d compared to the target distribution. Note that for some choices of f and n, not all $\perp$ entries will be overwritten in the die returned by Algorithm 1. For example, if f is the pmf of a weighted coin with $f(\text{“Heads”}) = 0.55$ and $f(\text{“Tails”}) = 0.45$, then for $d = 1DA(f,2)$, we have that $f_d(\text{“Heads”}) = 0.5$, $f_d(\text{“Tails”}) = 0$, and $f_d(\perp) = 0.5$. Lemma 2 demonstrates the tight relationship between the number of entries with label $\perp$ and the approximation error function ζ.

Lemma 2. *Given $d = 1DA(f,n)$ for an arbitrary integer $n > 0$ and an arbitrary probability mass function f with finite support, the probability mass that f_d allocates to $\perp$ is equal to the total additive approximation error of f_d. That is,*

$$f_d(\perp) = \zeta(f_d).$$

Proof: By the construction of Algorithm 1, we know that $n_\perp$, the number of entries in d with label $\perp$, is precisely defined by

$$n_\perp = n - \sum_{x \in supp(f)} \texttt{freq}_d(x)$$

where $\texttt{freq}_d(x)$ gives the number of times that x appears in the array d. Accordingly, we know that

$$\begin{aligned}
f_d(\perp) &= \frac{n_\perp}{n} \quad \text{by Definition 1} \\
&= \frac{n - \sum_{x \in supp(f)} \texttt{freq}_d(x)}{n} \\
&= \frac{n - \sum_{x \in supp(f)} \lfloor f(x) \cdot n \rfloor}{n} \quad \text{by Algorithm 1} \\
&= 1 - \sum_{x \in supp(f)} f_d(x) \quad \text{by Definition 1} \\
&= \sum_{x \in supp(f)} f(x) - \sum_{x \in supp(f)} f_d(x) \quad \text{by definition of pmf} \\
&= \zeta(f_d) \quad \text{by Fact 1.}
\end{aligned}$$

□

Error Bound for 1-Die Approximations. Now we are ready to prove Lemma 3 which bounds the total additive error when using Algorithm 1 to approximate an arbitrary discrete distribution with finite support.

Lemma 3. *Let f be a probability mass function whose support has at most n elements. Fix some integer $m > 1$. Let $d = 1DA(f, mn)$. Then we have*

$$\zeta(f_d) < \frac{1}{m}.$$

Proof: We have

$$\begin{aligned}
\zeta(f_d) &= \sum_{x \in supp(f)} f(x) - f_d(x) \quad \text{by Fact 1} \\
&= \sum_{x \in supp(f)} f(x) - \frac{\lfloor f(x) \cdot mn \rfloor}{mn} \quad \text{by Alg 1} \\
&= \sum_{x \in supp(f)} \frac{f(x) \cdot mn - \lfloor f(x) \cdot mn \rfloor}{mn} \\
&< \sum_{x \in supp(f)} \frac{1}{mn} \quad \text{since } \forall x \geq 0 \in \mathbb{R}, \quad x - \lfloor x \rfloor < 1 \\
&\leq \frac{1}{m}
\end{aligned}$$

where the last step follows since $|supp(f)| \leq n$ by our starting assumption. □

Lemma 3 implies that if we use a big enough table, we can achieve an approximation of any discrete, finite probability mass function with arbitrarily small error. However, exploiting this bound directly may require tables that are impractically large. We build on these results in the next section.

4.1.2 Approximating Distributions with Multiple Table Lookups

In this section we propose a method that composes several table lookups to achieve higher precision with greatly reduced computational overhead. Similarly to the previous section, we begin with a mathematical structure that precisely defines the usage of table lookups in a sampling process (Definition 3).

Definition 3. *A* **Dice Ensemble** $D = (V, E)$ *is a tree that encodes a discrete probability distribution with finite support. Each vertex* $v \in V$ *holds a die* d_v*, which is an array whose entries are either elements of a support set* $S_v \cup \{\perp\}$ *(where* $\perp$ *is a special element used to represent error when approximating a target distribution with support* S*), or a "placeholder" for another vertex* $u \in V$*. A directed edge exists from* v *to* u *if and only if* d_v *has a placeholder for* u*.*

To sample from the probability distribution encoded by D*, we use Algorithm 2. The probability mass function of this distribution can be computed using Algorithm 3.*

We note that while dice ensembles are defined as trees of dice, we only use unary 'trees' (i.e. each node possessing at most one child) in the present study. We leave this generality in the definition as it may allow for improved approximations via more complex decompositions of the target distribution than that of Algorithm 4. We reserve this possibility for future work.

Algorithm 2 DE-Sample (Sampling from a Dice Ensemble)

Input: a die ensemble $D = (V, E)$ with root vertex r
Output: an element from the set $\{\perp\} \cup \bigcup_{v \in V} S_v$

$X \leftarrow$ the set of leaves in V
while $X \neq \{r\}$ **do**
  for all $v \in X$ **do**
    $i \xleftarrow{\$} [n_v]$ where n_v is the number of entries in d_v
    $s \leftarrow d_v[i]$
    for all incoming edge to v, $(u, v) \in E$ **do**
      replace each placeholder v on d_u with s
      remove (u, v) from E
    end for
    remove v from V
  end for
  $X \leftarrow$ the set of leaves in V
end while
$i \xleftarrow{\$} [n_r]$ where n_r is the number of entries in the root die d_r
return $d_r[i]$

The sampling procedure Algorithm 2 rolls the dice at the leaves of the tree, putting the results into the placeholder spaces in the dice of parent nodes. The leaves are then removed. This repeats until only the root is left. Finally, the root die is rolled, and its output is the final result of the sampling algorithm. Algorithm 3 produces a probability mass function by leveraging the fact that the probability of obtaining a given value from any leaf vertex v is well defined (as it is simply a die). Accordingly, the probability that a placeholder for v in a parent node takes a given value is also well defined. Thus the iterative computation of probabilities from root to leaf described in the algorithm suffices to find the probability mass function of the distribution.

In Algorithm 4, we compose table lookups to approximate a distribution with increasing precision. For intuition, recall that in Algorithm 1, $f(x) \geq f_d(x) \forall x \in supp(f)$, and by Lemma 2, all of the probability mass comprising the difference between f and f_d is contained in entries of d that are labeled with the error symbol $\perp$. However, by construction of Algorithm 1, the gap between $f(x)$ and $f_d(x)$ is smaller than $\frac{1}{T}$ for all $x \in supp(f)$, which means that allocating an additional face of d to any $x \in supp(f)$ would cause $f_d(x) > f(x)$, and there would be no guarantee that the approximation error would decrease. However, if we divided up the mass from $f_d(\perp)$ into pieces smaller than $\frac{1}{T}$, we

Algorithm 3 DE-pmf (Probability Mass Function of a Dice Ensemble)

Input: a die ensemble $D = (V, E)$ with root vertex r

Output: a a dictionary representing a probability mass function g supported on $\{\perp\} \cup \bigcup_{v \in V} S_v$

$g \leftarrow$ a dictionary with key set $S \cup \{\perp\}$, values all initialized to 0
$w \leftarrow$ a dictionary with key set V, values all initialized to 0
$w[r] \leftarrow 1$
$X \leftarrow \{r\}$
while $X \neq \emptyset$ **do**
  $X' \leftarrow \emptyset$
  for all $v \in X$ **do**
    for all $i \in [0, |d_v|]$ **do**
      $s \leftarrow d_v[i]$
      if $s \in S \cup \{\perp\}$ **then**
        $g[s] \leftarrow g[s] + w[v] \cdot \frac{1}{|d_v|}$
      else
        $w[s] \leftarrow w[s] + \frac{1}{|d_v|}$
      end if
      $X' \leftarrow X' \cup \{\text{children of } v\}$
    end for
  end for
  $X \leftarrow X'$
end while
return g

could allocate it more productively to achieve a finer approximation. This will be our strategy in Algorithm 4.

Specifically, we obtain finer and finer approximations of the target distribution by constructing a 1-die approximation of the error distribution d_{i+1}, and assigning entries of d_i labeled with $\perp$ to take its outputs. Functionally, if we take f_{D_i} to be the probability mass function of the dice ensemble in the i^{th} iteration of Algorithm 4, this splits the approximation error $f_{D_i}(\perp)$ into pieces of size $\frac{f_{D_i}(\perp)}{T^{i+1}}$, and allocates them to fill the gaps between the target pmf and the approximation. This brings $f_{D_{i+1}}$ closer to f. This intuition is formalized in the proof of Theorem 2, our key theoretical result.

Theorem 2. *Let f be the probability mass function of an arbitrary discrete distribution with finite support. Let $D = DEA(f, \ell)$ be a dice ensemble obtained from Algorithm 4, and f_D be the probability mass function of the corresponding distribution. Then we have*

$$\sum_{x \in supp(f)} |f(x) - f_D(x)| < 2^{-\ell}.$$

The proof is deferred to Appendix A.1. Theorem 2 shows that if we use enough tables, we can achieve an arbitrarily fine approximation of any discrete distribution with finite support. Furthermore, the error between the approximated distribution and the target distribution given as input declines exponentially with the number of tables. It follows as a corollary that they are statistically indistinguishable when parameterized properly.

Algorithm 4 DEA (Dice Ensemble Approximation of a Distribution)

Input: A probability mass function $f : \mathbb{R} \mapsto [0, 1]$ corresponding to a finite discrete probability distribution with $|supp(f)| = n$; a maximum number of dice ℓ

Output: a dice ensemble D that approximates f

initialize $D = (V, E)$ with V, E empty sets
$Err \leftarrow$ a dictionary whose keys are $x \in supp(f)$, values initialized to $f(x)$
$c \leftarrow 1$ {tracks error mass}
for all $i \in [1, \ell]$ **do**
  $d_i \leftarrow \text{1DA}(Err, 2n)$
  for all $x \in supp(f)$ **do**
    $Err[x] \leftarrow Err[x] - \frac{\texttt{freq}_{d_i}(x)}{2n} \cdot c$
  end for
  $c \leftarrow c \cdot \frac{\texttt{freq}_{d_i}(\perp)}{2n}$
  $Err[x] \leftarrow \frac{Err[x]}{\sum_{x \in supp(f)} Err[x]} \quad \forall x \in supp(f)$ {re-normalize the error}
  $v_i \leftarrow$ vertex with d_i as its die
  $V \leftarrow V \cup \{v_i\}$
  if $i \neq 1$ **then**
    $E \leftarrow E \cup \{(v_i, v_{i-1})\}$
  end if
  if $f_{d_i}(\perp) = 0$ **then**
    break
  end if
  if $i < \ell$ **then**
    set all remaining entries of d_i with $\perp$ to v_{i+1} instead
  end if
end for
return $D = (V, E)$

4.1.3 Statistical Closeness to Input Distributions

In this section, we show that our constructed distributions are statistically indistinguishable from the target distributions they mimic. We then use a hybrid argument to extend this result, showing that any probabilistic algorithm which samples from an arbitrary discrete distribution X is statistically indistinguishable from an algorithm which uses our constructed distribution Y as a drop-in replacement.

Corollary 1 (Statistical Indistinguishability of Target Distribution and Table Approximation.)**.** *Consider an arbitrary discrete, finite random variable X with probability mass function f. Let $Y = (Y_\lambda)_{\lambda \in \mathbb{N}}$ be a sequence of random variables parameterized by a statistical security parameter λ, such that the probability mass function of $Y(1^\lambda)$ is given by f_{D_λ} where $D_\lambda = DEA(f, \lambda)$. Then Y is statistically indistinguishable from X.*

Proof: As an immediate consequence of Theorem 2 we have $\sum_{x \in supp(f)} |f(x) - f'(x)| \leq 2^{-\lambda}$. There is only one element in $supp(f')$ that is not in $supp(f)$, and that's $\perp$. So we have

$$
\begin{aligned}
SD_{X,Y}(\lambda) &= \frac{1}{2} \cdot \sum_{z \in \{0,1\}^*} \left| \Pr[X = z] - \Pr\left[Y(1^\lambda) = z\right] \right| \\
&\quad \text{(by def of Statistical Distance)} \\
&= \frac{1}{2} \cdot \left(f_{D_\lambda}(\perp) + \sum_{x \in supp(f)} |f(x) - f_{D_\lambda}(x)| \right) \\
&= \frac{1}{2} \cdot 2 \cdot \sum_{x \in supp(f)} |f(x) - f_{D_\lambda}(x)| \quad \text{by Lemma 2} \\
&\leq 2^{-\lambda} \quad \text{by Theorem 2.}
\end{aligned}
$$

Thus the statistical distance between X and Y is bounded by a negligible function of λ. □

Corollary 1 shows that we can approximate any discrete distribution with finite support finely enough to be indistinguishable using relatively few tables. To complement this result, we will define a truncation scheme which enables us to construct a statistically indistinguishable finitized version of any discrete distribution with a defined probability mass function in Algorithm 5.

Algorithm 5 `Trunc` (Statistically Close Truncation)

Input: f, the pmf of a discrete probability distribution; statistical security parameter λ
Output: f', a statistically close finitized pmf

1: $supp(f') \leftarrow \emptyset$
2: Let $(x_0, x_1, x_2, ...)$ be a sequence of values in $supp(f)$ such that $f(x_i) \geq f(x_{i+1}) \forall i \in \mathbb{N}$.
3: $i \leftarrow 0$
4: **while** $\frac{1}{2} \cdot \left(1 - \sum_{x \in supp(f')} f'(x)\right) < 2^{-\lambda}$ **do**
5: $\quad supp(f') \leftarrow supp(f') \cup \{x_i\}$
6: $\quad f'(x_i) \leftarrow f(x_i)$
7: $\quad i \leftarrow i + 1$
8: **end while**
9: return f'

Algorithm 5 straightforwardly defines a truncated probability mass function f' by copying over elements from f in descending order until the statistical distance between the two is underneath a

threshold defined by an exponential function of λ. This trivially results in f' which is statistically indistinguishable from f.

With Corollary 1 and Algorithm 5 together, we are ready to prove statistical indistinguishability between the outputs of a probabilistic algorithm which uses constantly-many samples from an arbitrary discrete distribution X (possibly with *infinite* support), and the same algorithm which samples from the MPC-efficient approximation given by $DEA(\texttt{Trunc}(f, \lambda), \lambda)$ as a drop-in replacement.

Theorem 3. *Consider an oracle O which outputs samples from a discrete distribution X. Consider also a probabilistic algorithm $\mathcal{A}$ which takes a database as input, makes a constant number $t \in \mathbb{N}$ of calls to O, and outputs a real number. Let O' be an oracle which outputs samples from the distribution defined by $DEA(\texttt{Trunc}(f, \lambda), \lambda)$, where f is the probability mass function of X. Let $\mathcal{A}'$ be a probabilistic algorithm that is exactly the same as $\mathcal{A}$, except the oracle calls to O are replaced by calls to O'.*

Then the output distributions of $\mathcal{A}$ and $\mathcal{A}'$ are statistically indistinguishable.

Proof: We proceed via a standard hybrid argument. Consider a sequence of hybrids $H^0, H^1, ..., H^t, H^{t+1}, ..., H^{2t}$, where H^0 is the distribution defined by the original mechanism M, and each hybrid up to H^t replaces a single call to O with a call to an oracle O'' which samples from $\texttt{Trunc}(f, \lambda)$, so that H^t is the same as M except that all calls to O are replaced with calls to O''. Further, let H^{t+1} through H^{2t} each replace a call of O'' with a call to O', so that H^{2t} is the distribution defined by the mechanism M'.

By the construction of Algorithm 5, each adjacent pair of hybrids H^i and H^{i+1} for $i \in [0, t]$ are statistically indistinguishable, and by Corollary 1 this also holds for $i \in [t, 2t]$. Further, the number of hybrids is a constant with reference to the security parameter λ. Thus the sum of their statistical distances is a negligible function of λ, and accordingly H^0 is statistically indistinguishable from H^{2t}. □

Relationship to Differential Privacy Of particular interest to our desired use case, Theorem 3 implies that the chance that an adversary can distinguish between a differentially private mechanism which uses our approximated distribution, and one that uses the original target distribution, is a negligible function of λ. Accordingly, as long as λ is set high enough the two mechanisms should be essentially interchangeable in practice. We note also that differentially private mechanisms are often subjected to similar perturbations when translating theory to practice. Implementations on finite computers are incapable of drawing from many theoretical distributions used routinely in differential privacy. Instead, they implicitly draw from statistically close finitized approximations.

This means that we can utilize our MPC-friendly distributions to achieve improved efficiency in differentially private collaborative learning applications. In the following section we detail our protocol for sampling from these distributions within MPC.

4.2 MPC Protocol for Sampling

We realize the large table lookups by first generating encryptions of one-hot vectors and computing a dot product of the table and the private one-hot vector. We first show how to generate the one-hot vectors in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model (Figure 2). Then, we describe the protocol to sample from a distribution using a chained table lookup.

The main idea behind generating one-hot vectors with minimal communication is to use the boolean to bitwise encryption functionality from Figure 2 to obtain the bitwise encryptions of a random value $r \leftarrow [2^\ell]$. Then, locally compute the encrypted one-hot vector such that it is zero in all places other than r. For instance if we have a $2 - bit$ vector and the random bits sampled are $b_0 = 0$ and $b_1 = 1$, then the one-hot vector can be generated as $[\bar{b_0}\bar{b_1}, b_0\bar{b_1}, \bar{b_0}b_1, b_0b_1] = [0010]$. Figure 3, describes the protocol in detail. Using the efficient one-hot vector generation protocol from

Functionality $\mathcal{F}_{\mathsf{ABB}}$

This functionality operates over a finite field $\mathbb{Z}_p$ (resp., $\mathbb{F}_2$) for arithmetic secret-shared values (resp., Boolean secret-shared values), and interacts with parties $P_1, \ldots, P_n$.

Random: Upon receiving $(\mathsf{Random}, \mathsf{type}, \mathsf{id})$ from all parties where $\mathsf{type} \in \{\mathsf{arith}, \mathsf{bool}\}$ and id is a fresh identifier, sample $r \leftarrow \mathbb{Z}_p$ or $r \leftarrow \{0,1\}$ relying on type, store $(\mathsf{id}, \mathsf{type}, r)$.

Boolean to Bitwise Encryption: Upon receiving $(\mathsf{B2E}, \mathbf{id}, c_0, c_1, \ldots, c_l)$ from all parties, where $(\mathbf{id}[i][j])$ for $i \in [0, M]$ and $j \in [0, \ell]$ are present in memory, retrieve $(\mathbf{id}[i], \mathsf{bool}, \mathbf{x}[i])$ for all $i \in [0, M]$ and store $(c_0, c_1, \ldots, c_l)$ where $c_j = [\![\mathbf{x}[0 \ldots M][j]]\!]$ for $j \in [0, \ell]$.

Encryption to Arithmetic shares: Upon receiving $(\mathsf{E2A}, \mathsf{id}, \mathsf{id}')$ from all parties where $(\mathsf{id}, \mathsf{enc})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{enc}, x)$ and store $(\mathsf{id}', \mathsf{arith}, x)$.

Output: Upon receiving $(\mathsf{Output}, P_i, \mathsf{type}, \mathsf{id})$ from all parties, where $(\mathsf{id}, \mathsf{type})$ is present in memory, retrieve $(\mathsf{id}, \mathsf{type}, x)$ and then output it to P_i.

Figure 2: Functionality for the MPC black box.

Protocol $\Pi_{\mathsf{one\text{-}hot}}$

Input: Parties $P_1, \ldots, P_n$ hold the following inputs:

- The set of public parameters pp for public-key BGV-THE.
- P_i holds a share of the secret key $\mathsf{sk}_i = s_i$ and the public key pk, M is the packing size.
- Size of the output vector 2^ℓ.

Generation of M encrypted one-hot vectors:

1. All parties call the (Random) command of $\mathcal{F}_{\mathsf{ABB}}$ to sample a vector of Boolean sharings $\langle R \rangle^b$ with $R \in \{0,1\}^{\ell * M}$.
2. All parties call the $(\mathsf{B2E}, \langle R \rangle^b)$ of the functionality $\mathcal{F}_{\mathsf{ABB}}$ to obtain bit-wise ciphertexts $c_0 = [\![R[0 \ldots M][0]]\!], c_1 = [\![R[0 \ldots M][1]]\!], \ldots, c_{l-1} = [\![R[0 \ldots M][\ell - 1]]\!]$, such that each encryption packs M values.
3. All parties compute $\hat{c}_i = [\![1]\!] - c_i$, for all $i \in [0, l-1]$, where $[\![1]\!]$ is encryption of a vector of size M with all 1 values.
4. Each party P_i sets $\mathsf{start_i} = 2^{\ell - \log(\mathsf{n}) + 1} * (\mathsf{i} - 1)$ and $\mathsf{end_i} = 2^{\ell - \log(\mathsf{n}) + 1} * \mathsf{i}$.
5. Each party P_i computes $[\![\mathsf{Prod[j]}]\!] = \prod_{\mathsf{k} \in [0, \mathsf{l}-1]} \begin{cases} c_k & \text{if } j[k] = 1 \\ \hat{c}_k & \text{otherwise} \end{cases}$ for all $j \in [\mathsf{start_i}, \mathsf{end_i}]$.

Output: Each party P_i outputs $c_j = [\![\mathsf{Prod[j]}]\!]$ for all $j \in [\mathsf{start_i}, \mathsf{end_i}]$.

Figure 3: Protocol to obtain encrypted one-hot vector.

Figure 3, it is easy to see how we can instantiate a chained table lookup. At each level, we have a table that uses the output from the previous level in some places. Since the parties have encryption shares of the private value, they first aggregate this to get the encryption of the result of sampling from the current table and they can use this ciphertext to locally compute the shares of the next dot-product as described in Figure 4.

Implementation optimizations. The naive generation of the one-hot vector requires $O(\ell * 2^{\ell})$ ciphertext-ciphertext multiplications. A simple way to reduce the computation per party is to divide the computation of the one-hot vector among all parties. Thus, each party P_i computes only $2^{\ell-\log(n)}$ values in the one-hot vector between $s_i = 2^{\ell-\log n+1}(i-1)$ and $e_i = 2^{\ell-\log n+1}i$. In Π_{noise}, we use the same s_i and e_i for P_i as $\Pi_{\mathsf{one-hot}}$.

Further, several intermediate products can be reused; thus, to minimize the computation overhead, each party follows the steps below:

1. Compute the common product value to be used for all the indices among $[s_i, e_i)$, i.e., set common places $x = s_i \oplus e_i \oplus (2^{\ell-1})$. For all ones in x,

$$c_{common} = \prod_{k\in[0,l-1]} \begin{cases} 1 & \text{if } x[k] \neq 1 \\ c_k & \text{if } s_i[k] = 1 \text{ and } x[k] = 1 \\ \hat{c}_k & \text{otherwise} \end{cases}$$

   This product can be computed as a reduction.

2. Each party starts with a two-dimensional array of ciphertexts denoted by V, where initially each column $V[i]$ has two ciphertexts $c_k, \hat{c_k}$ where $x[k] = 0$. If available, we multiply the first column with c_{common}.

3. Now similar to reduction, each party computes the tensor product of all the columns in V until only two columns remain in V. Note that we can go up to a single column; however, it requires the parties to store a much larger number of ciphertexts without any computational saving.

4. In steps 1c and 1d in Π_{noise} (Figure 4), the parties compute the corresponding ciphertext using V.

The above steps help reduce the depth of ciphertext-ciphertext multiplications, thus reducing the noise increase in the resulting ciphertext.

Implementation details. We implemented our secure noise sampling protocol using the $Func[ABB]$ (Figure 2) implementation from [17] which uses EMP-toolkit [31] and OpenFHE [2] as MPC and FHE frameworks respectively. The key-setup process is identical to [17] where parties generate their local secret keys and exchange the corresponding public keys, which are aggregated to be the final global public key. This cost has been included in the evaluation.

Proof of security. The security of $\Pi_{\mathsf{one-hot}}$ is intuitive because the parties do not communicate with each other in the $\mathcal{F}_{\mathsf{ABB}}$-hybrid model. Similarly, in Π_{noise}, parties only send the encryption shares of the dot product, which are indistinguishable from the encryption of random values given the security of the THE scheme.

Proof of correctness. We give a brief intuition of the correctness proof here. First, given $\mathcal{F}_{\mathsf{ABB}}$ steps 1- 3, lead to all parties obtaining the encryption of uniformly random ℓM bits and the negation of these bits. Step 5 computes the encryption of the one-hot vector values by computing the product of the ciphertexts corresponding to the bit decomposition of each index. The underlying message is 1 iff the random bits equal that of the index. Thus, the $\Pi_{\mathsf{one\text{-}hot}}$ results with each party P_i obtaining encryptions of random one-hot vector values for indices between s_i, e_i.

The protocol Π_{noise} involves obtaining the encrypted one-hot vector using $\Pi_{\mathsf{one-hot}}$ and then assuming a single table, we compute a dot product with the original table, thus given the randomness of the one-hot vector, the samples are drawn from random indices. For chained table-lookup, in

Protocol Π_{noise}

Inputs: Parties $P_1, \ldots, P_n$ hold the following inputs:

- The set of public parameters pp for public-key BGV-THE.
- P_i holds a share of the secret key $\mathsf{sk}_i = s_i$ and the public key pk, M is the packing size.
- All parties hold tables $T_0, \ldots, T_{\lambda-1}$ of length 2^ℓ. T_i for $i \in [1, \lambda-1]$ has public entries up to index x_i and $T_i[j] = \perp$ for all $j \in [x_i, 2^\ell - 1]$. $x_0 := 2^\ell$.

Generation of random samples:

1. For $i \in [\lambda]$ all parties follow the steps below:
   (a) All parties use $\Pi_{\mathsf{one-hot}}$ to obtain the encryption of N one-hot vectors of length 2^ℓ, such that P_j obtains $c_{s_j}, \ldots c_{e_j}$.
   (b) Each party P_j computes $c_p^j = \Sigma_{k=\max(s_j,x_i)}^{k=e_j} c_k$
   (c) Each party P_j computes $[\![y_i^j]\!] = \Sigma_{k=s_j}^{k=\min x_i, e_j} c_j T_i[j] + c_p^j [\![y_{i+1}]\!]$.
   (d) All parties send $[\![y_i^j]\!]$ to P_1. P_1 computes $[\![y_i]\!] = \Sigma_{j=1}^{j=n} [\![y_i^j]\!]$ and sends it to all other parties.
2. All parties call $(\mathsf{E2A}, [\![y_0]\!])$ of the functionality $\mathcal{F}_{\mathsf{ABB}}$ obtain $\langle \boldsymbol{y} \rangle^a$.

Output: Each party outputs additive shares of M random noise samples $\langle \boldsymbol{y} \rangle^a$.

Figure 4: Protocol for distributed noise sampling.

steps 1c, we use the result from the previous iteration for the empty $\perp$ values. Note that if the number of tables is high, we must bootstrap the ciphertext to ensure correctness. In step 1d, parties obtain the encryption of the random samples. In step 2, parties obtain the additive shares of the random noise sample using encryption to shares functionality from $\mathcal{F}_{\mathsf{ABB}}$.

4.3 Application of discrete Gaussian generation: DP-CL

The proposed noise sampling in MPC method can be integrated with FL to achieve DP guarantees under an *honest-but-curious* threat model. The clients follow the protocol to compute local gradient updates and the server party will use secure MPC to aggregate all updates shared by the client parties. The appropriate amount of noise is sampled and added inside the secure MPC protocol. This noise is unknown to both the clients and the server. Hence there is no need to trust the clients or the server or add additional noise due to potentially colluding clients. In this work, we primarily consider applications like hospital collaboration where data point-level privacy shall be protected[1].

Our proposed framework is built upon two prior works [14] (with data point-level DP) and [20] (with client-level DP), which both use distributed DP. As shown in Algorithm 6 and Algorithm 7, in this work, we replace the distributed noise with central noise sampled inside the secure MPC protocol. We follow the same sampling and training process as in [14] to achieve data-point level privacy protection: we assume there are N clients and the size of their local dataset D_i is public. They all agree on the set of hyperparameters for model training as well as privacy budget ϵ before the training starts. For each communication round, all client parties are sampled to train and share a model update. All clients use the same mini-batch subsampling rate to draw data points for training their local model updates and each client computes, clips, and discretizes point-wise

[1]The proposed noise sampling protocol can also be used for client-level privacy protection, which is demonstrated in Appendix D.

gradient updates. The model updates are aggregated after each local training step. In addition, we follow the `conditional randomized rounding`, `flattening`, and `modular clipping` steps, in [20] to discretize the clients' model updates before secure MPC protocol, as defined in Appendix C. In addition, following the proofs in [20], we provide a similar utility and privacy analysis for our DP-CL method in Appendix C. A more detailed version of the algorithms is provided in Appendix B.

By Theorem 1, one single communication round of Algorithm 6 and Algorithm 7 satisfy $\frac{1}{2}\epsilon^2$-concentrated DP, with

$$\Delta_2^2 = \min \left\{ \begin{array}{c} c^2 + \frac{1}{4}\gamma^2 d + \sqrt{2\log(1/\beta)} \cdot \gamma \cdot \left(c + \frac{1}{2}\gamma\sqrt{d}\right), \\ \left(c + \gamma\sqrt{d}\right)^2 \end{array} \right\},$$

with $\epsilon = \frac{\Delta}{\sigma}$ when sampling probability $p = 1.0$. We follow the same privacy accounting methods [13, 3, 27, 32] (to handle subsampling and composition) used in [20] for consistency.

Algorithm 6 Client Training Procedure

Input:

- the current model state W
- the loss function L
- the clipping norm, c
- the noise multiplier, σ
- the discretization granularity, γ
- the bias β
- the mini-batch sampling rate, p
- the private training data D_i for client i,
- m for modulus operation
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$

1: Sample mini-batch $B_i \in D_i$ with sampling probability p.
2: **for all** $x_b \in B_i$ **do**
3: $g_b \leftarrow \nabla_W L(W, x_b)$ {compute example-wise gradient}
4: $g_b{}' \leftarrow \frac{1}{\gamma}(g_b \cdot \min\{1, \frac{C}{||g_b||_2}\})$ {clip and **scale** the gradient}
5: $g_b{}'' \leftarrow \texttt{flatten}(g_b{}', \xi)$
6: $\tilde{g_b} = \texttt{randomized round}(g_b{}'')$ {condition on $||\tilde{g_b}||_2 \leq \min\left\{\frac{c}{\gamma} + \sqrt{d}, \sqrt{\frac{c^2}{\gamma^2} + \frac{1}{4}d + \sqrt{2\log\frac{1}{\beta}} \cdot (\frac{c}{\gamma} + \frac{1}{2}\sqrt{d})}\right\}$}
7: **end for**
8: $\Delta W_i = \sum_b \tilde{g_b}$
9: output ΔW_i mod m for secure MPC

5 Empirical Evaluation

5.1 Summary of Evaluation

Below we summarize the key findings from our evaluation:

Algorithm 7 Server Training Procedure

Input:

- the previous model state W_{t-1}
- the noise multiplier, σ
- the discretization granularity, γ
- m for modulus operation
- the client dataset sizes $||D_i||$ (assumed to be public).
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$
- the models updates $\overline{\Delta W} = ((\sum_i \Delta W_i' \bmod m) + (\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2) \bmod m)) \bmod m$ via secure MPC $\{$this is equivalent to aggregating all scaled, flattened, and rounded gradient vectors: $= \left(\sum_b^B \bar{g}_b + (\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2))\right) \bmod m\}$

1: Map $\overline{\Delta W}$ to $\overline{\Delta W}' \in [-m/2, m/2]^d \cap \mathbb{Z}^d$ $\{$make $\overline{\Delta W}' \bmod m = \overline{\Delta W}\}$
2: $\Delta W = \gamma \cdot \texttt{unflatten}(\overline{\Delta W}', \xi)$
3: $B = p \cdot \sum_i^N ||D_i||$ {Calculate the overall mini-batch size }
4: output $W_t = W_{t-1} +$ optimizer$(\Delta W/B)$

- Our protocol achieves massive run time improvement when compared to prior state-of-the-art, namely, 450× for 32 parties.
- For communication cost, we require significantly less communication compared to the prior work, namely, 13× for 32 parties.
- The running time of our protocol is largely independent of the table size and statistical security parameter dominates the cost the protocol.
- The DP-CL algorithm that samples and adds noise via an MPC protocol achieves better model performance than algorithms that use distributed DP and need to add extra noise to account for colluding clients.

5.2 Evaluation Setup

Secure sampling experiments setup. All experiments are conducted on AWS of instance type `m5.2xlarge`. We benchmark our protocol for up to 32 parties to show the scalability of our protocol, and to support future applications which may require generating DP noise over larger numbers of parties. We consider the following network settings:

- **LAN:** Bandwidth of up to 1 Gbps and a latency of 0.1 ms.
- **WAN:** Bandwidth of up to 1 Gbps and a latency of 100 ms.

We compare our work with the prior state-of-the-art work by Wei et al. [34] for semi-honest security and up to $n-1$ corruptions. For comparison, we benchmark their protocol[2] using the GMW protocol from MP-SPDZ for semi-honest security. Specifically, we use OT-based `semi2k` and HE-based `temi`

[2]https://github.com/yuchengxj/Secure-sampling-benchmark

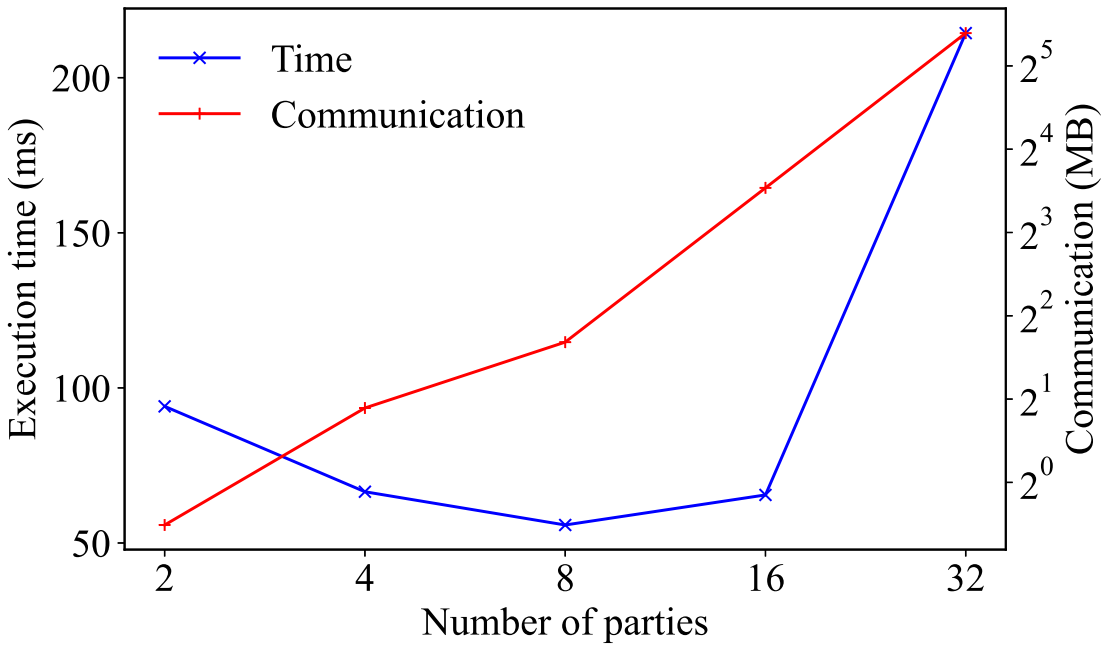

Figure 5: **Comparison for different number of parties.** Performance comparison for different number of parties with table size 2^{16} and statistical security parameter 64 in the LAN setting.

protocols from MPSPDZ to benchmark the prior work. All our experiments use bit-length equals 32 and statistical security parameter $\lambda = 64$ unless stated otherwise.

DP-CL experiments setup. To evaluate model utility, we simulate our DP-CL method for training soft prompts and the classifier layers of RoBERTa-base models [23] using standard natural language processing datasets SST2 [29], QQP [33], MNLI [35], and QNLI [30] from the GLUE benchmark [30] following prior works on DP fine-tuning of language models [7]. For all experiments, we simulate the distributed setting by splitting the dataset equally into 10 clients. Since we primarily aim for applications like hospital collaborations, the number of clients is not too big and is comparable to the numbers used in prior work [14]. Data point-level DP is considered and the δ value for DP is set to be $1/||D_{\text{train}}||$, where $||D_{\text{train}}||$ is the size of aggregated training data (summation of all clients' local data). The code is implemented in PyTorch. We followed the implementations of prior work [20] to discretize model gradients, simulate discrete Gaussian noise, and perform DP analysis. More details about the hyperparameters for training and discretization are described in Table 4 and Appendix E. All DP-CL experiments are conducted on NVIDIA A100 GPUs.

5.3 Efficiency of DP Noise Sampling

Scalability with number of parties. Figure 5 describes detailed performance of our protocol for increasing number of parties with statistical security parameter 64. We observe that the time decreases linearly up to 8 parties but increases for larger number of parties. This is because the runtime can be broken into two parts: (A) interactive part (computing encrypted bits from random boolean shares and additive shares of the output from its ciphertext), and (B) local computation (one-hot vector generation and output value encryption) (see Section 4.2). The local computation is parallelized across all parties, thus *decreasing* the runtime as the number of parties increases. At greater than 8 parties, the interactive part begins to dominate the runtime, resulting in the trend observed in Figure 5.

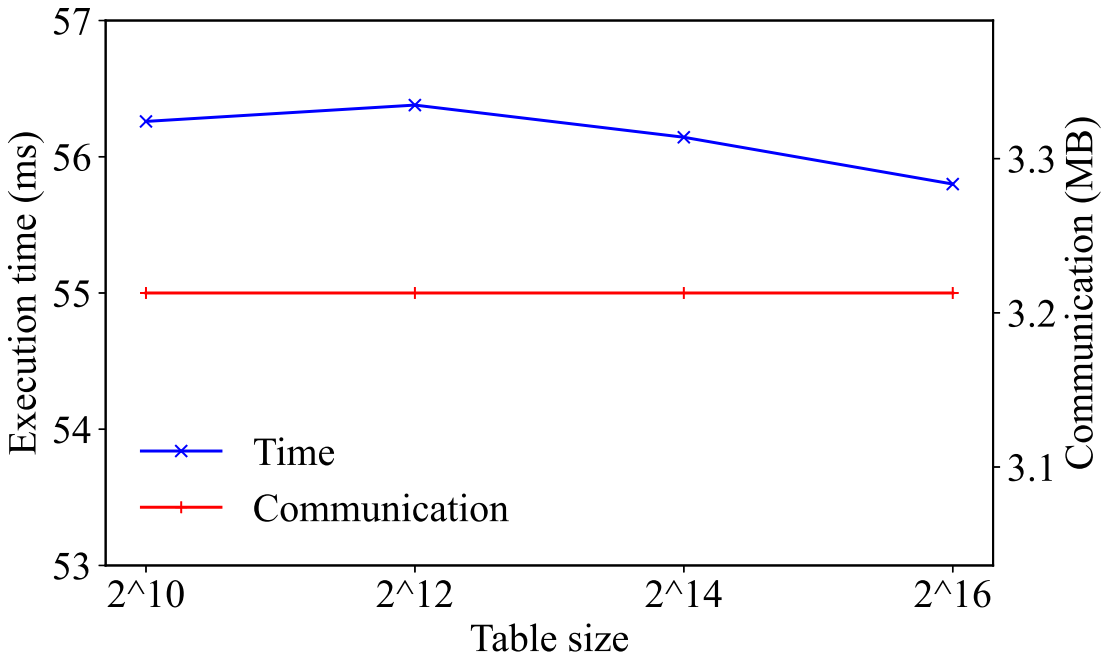

Figure 6: **Comparison for different table sizes.** Performance comparison for different table sizes for 8 parties with 64 statistical security parameter in the LAN setting.

Table 2: **Performance for varying statistical security parameter (λ).** These benchmarks are for 16 parties when using tables of size 2^{16} in the LAN setting.

λ	Time (ms)	Communication (MB)
40	40.53	7.21
64	64.42	11.18
128	128.09	22.22

Scalability with size of table. Figure 6 shows the performance using 64 tables/statistical security parameters and 8 parties. We note that communication is almost constant for any table size; this is because only the initial step of converting random indices to their bitwise encryption depends on the log of table size, and it is much smaller than the number of tables. As a result, communication largely remains independent of table size. Additionally, we note that computation time remains nearly constant with increasing table size. This is because most of the computation that depends on table size can be parallelized. Thus, the statistical security parameter becomes the dominant factor influencing performance.

Scalability with statistical security parameter. We benchmark the performance of our protocol for varying statistical security parameter with 16 parties and table size 2^{16} in Table 2. As noted earlier the computation dependent on λ is not parallelized, thus we observe that the performance depends linearly on λ. These experiments show that our method maintains efficiency even at minuscule levels of statistical distance from the target distribution. Further, Figure 7 illustrates how table size varies with the standard deviation σ for the discrete Gaussian distribution. We observe that for our CL benchmarks the table sizes are within the range of 2^{12} to 2^{16}, thus justifying our benchmark table sizes.

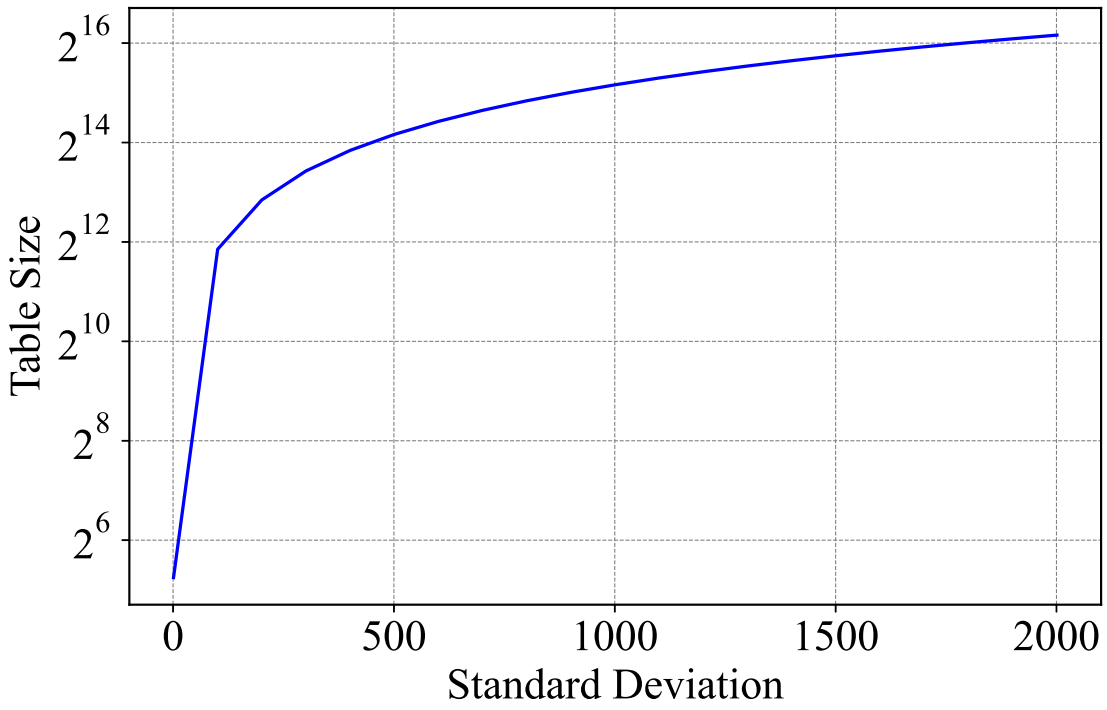

Figure 7: **Required table size vs standard deviation for the discrete Gaussian distribution. Comparison with prior work.** We show a detailed comparison of our work with the prior state of the art for different numbers of parties in Table 1. As shown in the table our protocol achieves significant improvement both in terms of the execution time and total communication required. For instance, when using 16 parties, our protocol is up to 213× faster and requires 2.8× less communication than prior work with HE-based protocol in the LAN setting. Note that we use the worst-case communication; as in the chained table lookup, the intermediate results are only used by a few parties; however, in our benchmarks, we send the intermediate results to all parties to keep the result independent of the underlying distribution. We extrapolate the cost of the prior work for the WAN setting based on the linear dependence of time vs latency as shown in Figure 9. Note that we use the implementation[3] for the prior work without any changes.

5.4 Model Utility Evaluation

CL with data point-level privacy guarantee by addition of centralized noise sampled in secure MPC. As a baseline, we first evaluate DP-CL without any discretization; the additive noise is continuous Gaussian sampled at the server side. We evaluate our method which discretizes the clients' gradient updates and adds the centralized discrete Gaussian noise as if it is sampled from a secure MPC protocol. We additionally show two DP-CL scenarios where distributed noise is added by the clients but with half of the clients or all but one client colluding. In our setup with 10 clients in total, those are equivalent to 5 and 9 colluding clients respectively. In those two cases, each client samples noises as if only $(10 - \text{num\_colluding})$ client(s) will contribute to the additive noise. That means, the total amount of noise added will be more than the amount of noise added in the centralized case, which may degrade the model performance more. The results evaluated on the SST2 dataset is plotted in Figure 8. We execute five runs for each experiment and include the 95% confidence interval and the mean ± standard deviation (Table 5). With half of the clients colluding, the final evaluation accuracy value is close to that of ours, but it converges more slowly than ours. When 9 clients collude, the training converges significantly more slowly and the final

[3]https://github.com/yuchengxj/Secure-sampling-benchmark

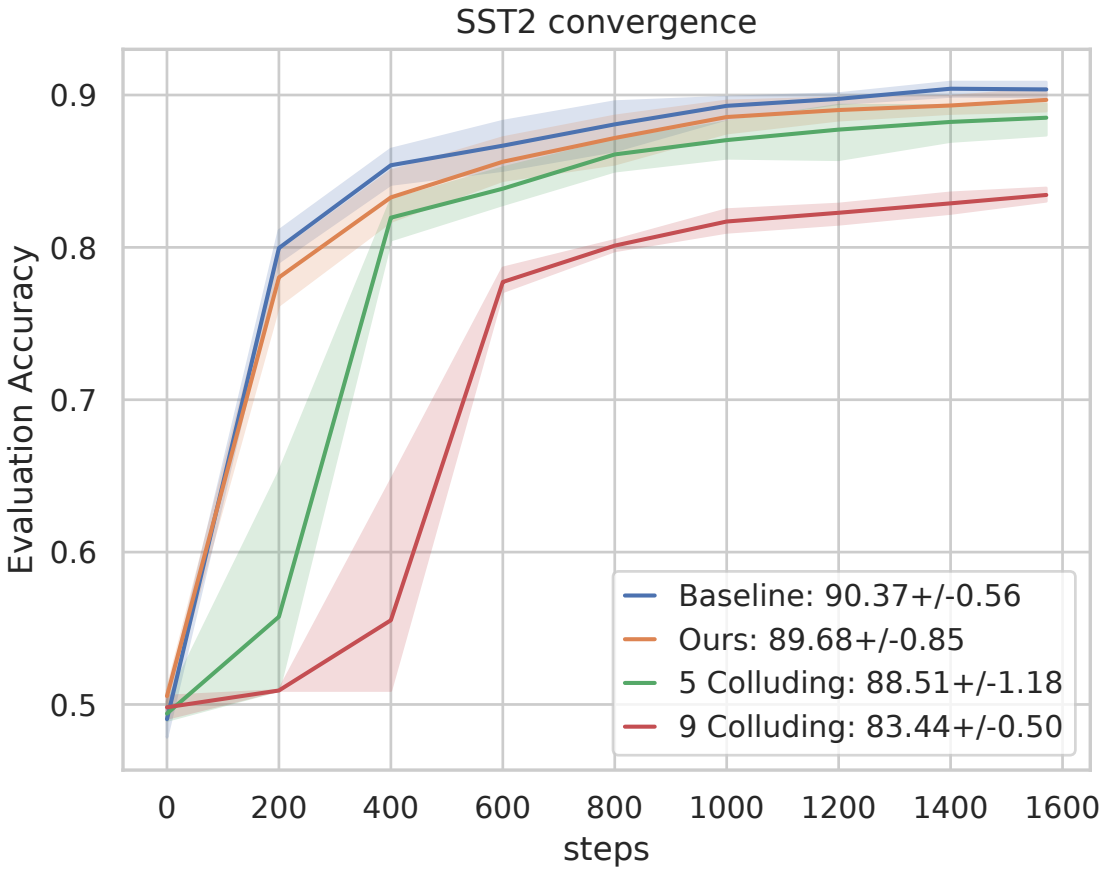

Figure 8: **The convergence plots on SST2 dataset.** Baseline shows the results for DP-FL with central continuous Gaussian sampled at the server side (without discretization). For Ours, we discretize the clients' gradient updates and add the centralized discrete Gaussian noise as if it is sampled from the secure MPC protocol. In addition, we show the performance of DP-FL with distributed noises but with 5 or 9 out of 10 clients colluding. In such cases, each client samples and adds noises as if only $(10 - \text{num\_colluding})$ client(s) will contribute to the additive noise. This leads to more total noise added during the training than for the centralized case, which degrades the model performance more. We run every experiment five times and plot the 95% confidence interval. In the legend, we report the final evaluation accuracy values in the format of mean $\pm$ std. In general, the cases with more colluding clients lead to lower evaluation accuracy values and slower convergence. Ours has slightly slower convergence and final evaluation accuracy than those of the Baseline method, but comparable.

evaluation accuracy is much lower than ours. A similar trend is observed for experiments on MNLI datasets (see Figure 12 in Appendix F.2). For QQP and QNLI datasets, the proposed method slightly outperforms the baseline, as shown in Figures 11 and 10. This is likely because our method requires additional discretization steps, which should slightly degrade the model's performance; however, our method also uses discrete noise, which provides comparable or slightly better accuracy, as noted in [4]. Depending on the relative contributions of these factors, and the stochasticity of model training, our method may perform slightly better or worse than the baseline.

Overall, ours performs similarly to the baseline (the mean values are similar and the confidence intervals overlap). On average, the method with 5 or 9 colluding parties performs worse than the

Table 3: End-to-end accuracy improvement vs. computational overhead for DP collaborative learning with 10 parties on 4 datasets. We show the improvement in mean model accuracy (Δ Acc) of our method compared to distributed DP with noise calibrated to protect from 9 and 5 colluding clients respectively (note that our method gives protection against all-but-one corruption, equivalent to the 9 colluding case). We also report runtime (WAN setting) and communication overhead of MPC noise sampling per training round.

Dataset	Δ Acc (9 collude)	Δ Acc (5 collude)	Time (s/round)	Communication (GB/round)
SST2	6.24	1.17	610	25.01
QNLI	3.04	0.51	666	27.29
MNLI	17.68	1.64	721	29.56
QQP	6.26	1.97	500	20.47

baseline or our method. This shows that by integrating DP collaborative ML training frameworks into secure MPC protocols, there is little degradation compared to cases without secure MPC protocols. By sampling noises in the secure MPC protocol, our method avoids adding additional noises to account for colluding clients, which yields better model performance.

We summarize the utility evaluation and contextualize it with the overhead of MPC noise sampling in Table 3. Our method provides protection against all-but-one corruption, which is equivalent to the 9 colluding setting for the distributed DP experiments. MPC sampling allows us to add less total noise to provide this level of protection. As a result, our method obtains large to moderate accuracy gains over distributed DP with protection against 9 colluding parties (we improve mean accuracy between 17.68% and 3.04%). By relaxing protections against colluding parties, distributed DP can achieve accuracy closer to our method, as seen in the 5 colluding setting experiments (we improve mean accuracy between 1.97% and 0.51%). While our sampling method improves significantly on computational overhead compared to the previous work (Table 1), the computational cost of MPC sampling aggregated over many model parameters remains substantial (requiring 500-721 seconds and 20.47-29.56 GB communication per training round). Thus, while our method achieves the highest level of accuracy and collusion protection, the accuracy/overhead tradeoff is substantially better in settings that demand strict privacy guarantees against many colluding parties. This underscores the importance of further improving the efficiency in future work, to make MPC sampling practical in a wider range of contexts.

6 Conclusions

In this work, we proposed a novel method for MPC noise sampling, tailored for the collaborative learning setting. Our method takes an arbitrary discrete distribution X as input, and finds a statistically indistinguishable approximation of X which can be sampled via a series of table lookups. This allows it to be sampled via our highly optimized protocol for uniform random table lookups in the semi-honest model. This approach is a departure from previous methods for MPC noise sampling, which work 'top-down' to adapt sampling algorithms for particular distributions of interest to the MPC setting. Our 'bottom-up' design, which generically adapts distributions of interest so that they are suitable for our MPC sampling method, results in improved efficiency and flexibility. Through our experiments, we show that our novel protocol improves substantially in terms of runtime and communication in comparison to prior works, especially at higher number of participating parties. We provide proofs that the noise sampled using our protocol is statistically indistinguishable to the distributions ingested as input. In addition, we conduct an end-to-end simulation of using the proposed MPC noise sampling method in collaborative training.

Limitations and Future Work. Our generic design pattern offers several opportunities for extensions and additional applications.

- *Security Model.* It may be of interest to perform noise generation in the malicious security model rather than semi-honest. Exploring application of malicious-secure MPC or zero-knowledge proof methods (e.g. [36]) for table lookups is a promising area for future work.
- *Continuous Distributions.* Understanding how to apply our method to continuous noise distributions may also be of interest. An idea similar to the snapping mechanism of [26] may provide a general and theoretically sound way to make continuous noise distributions suitable for our sampling method.
- *Further Improvements to Overhead.* While our sampling method improves computational costs substantially compared to previous work, it still results in high runtime and communication per training round. Further efficiency improvements are necessary to make MPC sampling practical in many contexts. More precise theoretical tools for decomposing target distributions into table lookups, and/or characterizing a relationship between approximation fineness and DP parameters rather than using statistically indistinguishable noise, may enable construction of more efficient samplers in future work.

Acknowledgments

Work of Xiao Wang was supported by NSF #2236819. Adam Dziedzic is supported by the the OpenAI Cybersecurity Grant. Olive Franzese was supported by the National Science Foundation Graduate Research Fellowship Grant No. DGE-1842165. We would like to also acknowledge our other sponsors, who support our research with financial and in-kind contributions: Amazon, Apple, CIFAR through the Canada CIFAR AI Chair, Meta, NSERC through the Discovery Grant and an Alliance Grant with ServiceNow and DRDC, the Ontario Early Researcher Award, the Schmidt Sciences foundation through the AI2050 Early Career Fellow program, and the Sloan Foundation. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *ACM CCS 2016*, 2016.

[2] Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. `https://eprint.iacr.org/2022/915`.

[3] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *TCC 2016*, 2016.

[4] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. The discrete Gaussian for differential privacy. In *NIPS*, 2020.

[5] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. Securely sampling biased coins with applications to differential privacy. In *CCS*, 2019.

[6] Christopher A Choquette-Choo, Natalie Dullerud, Adam Dziedzic, Yunxiang Zhang, Somesh Jha, Nicolas Papernot, and Xiao Wang. Capc learning: Confidential and private collaborative learning. In *ICLR*, 2021.

[7] Haonan Duan, Adam Dziedzic, Nicolas Papernot, and Franziska Boenisch. Flocks of stochastic parrots: Differentially private prompt learning for large language models. In *NeurIPS 2023*, 2023.

[8] Cynthia Dwork. Differential privacy. In *ICALP*, 2006.

[9] Cynthia Dwork. A firm foundation for private data analysis. *Commun. ACM*, 2011.

[10] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *EUROCRYPT*, 2006.

[11] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[12] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. 2014.

[13] Cynthia Dwork and Guy N Rothblum. Concentrated differential privacy. 2016.

[14] Congyu Fang, Adam Dziedzic, Lin Zhang, Laura Oliva, Amol Verma, Fahad Razak, Nicolas Papernot, and Bo Wang. Decentralised, collaborative, and privacy-preserving machine learning for multi-hospital data. *eBioMedicine*, 2024.

[15] Marc Fischlin and Arno Mittelbach. An overview of the hybrid argument. Cryptology ePrint Archive, Paper 2021/088, 2021.

[16] Yucheng Fu and Tianhao Wang. Benchmarking secure sampling protocols for differential privacy. In *ACM CCS*, 2024.

[17] Radhika Garg, Kang Yang, Jonathan Katz, and Xiao Wang. Scalable mixed-mode mpc. In *IEEE S&P*, 2024.

[18] Robin C. Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. *CoRR*, 2017.

[19] Mikko Heikkilä, Eemil Lagerspetz, Samuel Kaski, Kana Shimizu, Sasu Tarkoma, and Antti Honkela. Differentially private bayesian learning on distributed data. In *NeurIPS*, 2017.

[20] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *ICML*, 2021.

[21] Georgios Kaissis, Alexander Ziller, Jonathan Passerat-Palmbach, Théo Ryffel, Dmitrii Usynin, Andrew Trask, Ionésio Lima, Jason Mancuso, Friederike Jungmann, Marc-Matthias Steinborn, Andreas Saleh, Marcus Makowski, Daniel Rueckert, and Rickmer Braren. End-to-end privacy preserving deep learning on multi-institutional medical imaging. *Nature Machine Intelligence*, 2021.

[22] Hannah Keller, Helen Möllering, Thomas Schneider, Oleksandr Tkachenko, and Liang Zhao. Secure noise sampling for dp in mpc with finite precision. In *ACM ARES*, 2024.

[23] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019.

[24] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *AISTATS*, 2017.

[25] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. In *ICLR*, 2018.

[26] Ilya Mironov. On significance of the least significant bits for differential privacy. In *ACM CCS*, 2012.

[27] Ilya Mironov. Rényi differential privacy. In *IEEE CSF*, 2017.

[28] Nicolas Papernot, Shuang Song, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Ulfar Erlingsson. Scalable private learning with pate. In *ICLR*, 2022.

[29] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *ACL EMNLP*, 2013.

[30] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *ICLR*, 2019.

[31] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. EMP-toolkit: Efficient MultiParty computation toolkit. `https://github.com/emp-toolkit`, 2016.

[32] Yu-Xiang Wang, Borja Balle, and Shiva Prasad Kasiviswanathan. Subsampled renyi differential privacy and analytical moments accountant. In *AISTATS*, 2019.

[33] Zhiguo Wang, Wael Hamza, and Radu Florian. Bilateral multi-perspective matching for natural language sentences. In *IJCAI 2017*.

[34] Chengkun Wei, Ruijing Yu, Yuan Fan, Wenzhi Chen, and Tianhao Wang. Securely sampling discrete gaussian noise for multi-party differential privacy. In *ACM 2023*, 2023.

[35] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL-HLT 2018*, 2018.

[36] Kang Yang, Pratik Sarkar, Chenkai Weng, and Xiao Wang. Quicksilver: Efficient and affordable zero-knowledge proofs for circuits and polynomials over any field. In *ACM CCS*, 2021.

A Supplementary Methods

A.1 Proof of Main Theorem

Theorem 4 ((Reiteration of Main Theorem))**.** *Let f be the probability mass function of an arbitrary discrete distribution with finite support. Let $D = DEA(f, \ell)$ be a dice ensemble obtained from Algorithm 4, and f_D be the probability mass function of the corresponding distribution. Then we have*

$$\sum_{x \in supp(f)} |f(x) - f_D(x)| < 2^{-\ell}.$$

Proof:

Fix an arbitrary discrete distribution with finite support, and let f be its probability mass function. Let $(D_1, D_2, D_3, ...)$ be a sequence of dice ensembles defined by $D_i := DEA(f, i)$, and let f_{D_i} be the corresponding probability mass function of each. Let $d_i := 1DA(Err, 2 \cdot |supp(f)|)$ as in line 5 of Algorithm 4 in its i^{th} iteration, and let f_{d_i} be the corresponding probability mass function of this die.

Define error mass functions $f_{E_1}(x) := f(x) \quad \forall x \in supp(f)$ and $f_{E_n}(x) := f(x) - f_{D_{n-1}}(x) \quad \forall n > 1 \in \mathbb{N}$. For any integer $n > 0$, define normalized forms of the error mass functions $\bar{f}_{E_n}(x) := \frac{f_{E_n}(x)}{\sum_{x \in supp(f)} f_{E_n}(x)}$.

We begin the proof by pointing out two Facts which will be useful later.

Fact A: note that the following equality is implied by the construction of Algorithm 4

$$f_{D_n}(x) = f_{D_{n-1}}(x) + f_{D_{n-1}}(\perp) \cdot f_{d_n}(x)$$

since for every die in the ensemble besides the one in the leaf node, the $\perp$ symbols written on d_{i-1} get replaced with placeholders for the result of rolling d_i.

Fact B: Observe that as in the corresponding Lemma 2 for 1-die approximations, the probability mass occupied by $\perp$ in the dice ensemble has a tight relationship with the error function f_{E_n}.

$$\begin{aligned}
f_{D_{n-1}}(\perp) &= 1 - \sum_{x \in supp(f)} f_{D_{n-1}}(x) \quad \text{by construction} \\
&= \sum_{x \in supp(f)} f(x) - \sum_{x \in supp(f)} f_{D_{n-1}}(x) \quad \text{since } f \text{ is a pmf} \\
&= \sum_{x \in supp(f)} f(x) - f_{D_{n-1}}(x) \\
&= \sum_{x \in supp(f)} f_{D_{n-1}}(x) + f_{E_n}(x) - f_{D_{n-1}}(x) \quad \text{by def of } f_{E_n} \\
&= \sum_{x \in supp(f)} f_{E_n}(x).
\end{aligned}$$

Accordingly, we know that $\frac{f_{E_n}(x)}{f_{D_{n-1}}(\perp)} = \bar{f}_{E_n}(x)$.

Now we are ready to proceed to proving the desired result by induction.

The base case is given by $\sum_{x \in supp(f)} f(x) - f_{D_1} < \frac{1}{2}$ which follows directly by Lemma 3, since $D_1 = d_1$ is a 1-die approximation of $f(x)$.

For the inductive step, assume

$$\sum_{x \in supp(f)} f(x) - f_{D_{i-1}} < 2^{-(i-1)}.$$

Fact C: This implies $f_{D_{i-1}}(\perp) < 2^{-(i-1)}$ by an argument trivially similar to Fact B.

Then we have

$$\sum_{x \in supp(f)} f(x) - f_{D_i}(x)$$

$$= \sum_{x \in supp(f)} f_{D_{i-1}}(x) + f_{E_i}(x) - f_{D_i}(x) \quad \text{by def of } f_{E_i}$$

$$= \sum_{x \in supp(f)} f_{D_{i-1}}(x) + f_{E_i}(x) - (f_{D_{i-1}}(x) + f_{D_{i-1}}(\perp) \cdot f_{d_i}(x))$$

by Fact A

$$= \sum_{x \in supp(f)} f_{E_i}(x) - f_{D_{i-1}}(\perp) \cdot f_{d_i}(x)$$

$$= \sum_{x \in supp(f)} f_{D_{i-1}}(\perp) \cdot \frac{f_{E_i}(x)}{f_{D_{i-1}}(\perp)} - f_{D_{i-1}}(\perp) \cdot f_{d_i}(x)$$

multiplying by 1

$$= \sum_{x \in supp(f)} f_{D_{i-1}}(\perp) \cdot \bar{f}_{E_i}(x) - f_{D_{i-1}}(\perp) \cdot f_{d_i}(x)$$

by Fact B

$$= f_{D_{i-1}}(\perp) \sum_{x \in supp(f)} \bar{f}_{E_i}(x) - f_{d_i}(x)$$

$$< f_{D_{i-1}}(\perp) \cdot \frac{1}{2}$$

since d_i is a 1-die approx of $\bar{f}_{E_i}$ by construction, and then by Lemma 3

$$< 2^{-(i-1)} \cdot \frac{1}{2} \quad \text{by inductive assumption and Fact C}$$

$$= 2^{-i}.$$

So the inductive step holds. □

B For per-data point privacy guarantee

Detailed training algorithm for the clients and server for data point-level privacy is outlined in Algorithms 8 and 9. Each client has local dataset D_i for $i \in [N]$ and the dataset sizes $||D_i||$ are publicly known. The clients and the server agreed on the set of hyperparameters, e.g., the loss function L, the clipping norm, c, the noise multiplier, σ, the discretization granularity, γ, the bias for rounding, β, the mini-batch sampling rate, p, m for modulus operation. For each iteration, there is a public uniformly random sign vector $\xi \in \{-1, +1\}^d$ useful for flattening the gradient vectors.

C Definition and Notation

We leverage the following definition from [20], which is originally designed for FL with distributed DP guarantee. We also perform a similar utility and privacy analysis here for our centralized noise addition inside secure MPC protocol. The differences are highlighted in red.

Definition 4 (Randomized Rounding [20])**.** *Let $\gamma > 0$ and $d \in \mathbb{N}$. Define $R_\gamma : \mathbb{R}^d \to \gamma\mathbb{Z}^d$ (where $\gamma\mathbb{Z}^d := \{(\gamma z_1, \gamma z_2, \cdots, \gamma z_d) : z_1, \cdots, z_d \in \mathbb{Z}\} \subset \mathbb{R}^d$) as follows. For $x \in [0, \gamma]^d$, $R_\gamma(x)$ is a product distribution on $\{0, \gamma\}^d$ with mean x; that is, indepdently for each $i \in [d]$, we have $\mathbb{P}[R_\gamma(x)_i = 0] =$*

Algorithm 8 Client Training Procedure (detailed)

Input:

- the current model state W
- the loss function L
- the clipping norm, c
- the noise multiplier, σ
- the discretization granularity, γ
- the bias β
- the mini-batch sampling rate, p
- the private training data D_i for client i,
- m for modulus operation
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$

Sample mini-batch $B_i \in D_i$ with sampling probability p.
for all $x_b \in B_i$ **do**
  $g_b \leftarrow \nabla_W L(W, x_b)$ {compute example-wise gradient}
  $g_b' \leftarrow \frac{1}{\gamma}(g_b \cdot \min\{1, \frac{C}{||g_b||_2}\})$ {clip and **scale** the gradient}
  $g_b'' \leftarrow H_d D_\xi g_b'$ where $H \in \{-1/\sqrt{d}, +1/\sqrt{d}\}$ is a Walsha-Hadamard matrix satisfying $H^T H = I$ and $D_\xi \in \{-1, 0, +1\}^{d \times d}$ is a diagonal matrix with ξ on the diagonal {**flatten** the gradient}
  repeat $\tilde{g_b} = \texttt{randomized round}(g_b'')$ until $||\tilde{g_b}||_2 \leq \min\left\{c/\gamma + \sqrt{d}, \sqrt{c^2/\gamma^2 + \frac{1}{4}d + \sqrt{2\log 1/\beta} \cdot (c/\gamma + \frac{1}{2}\sqrt{d})}\right\}$ {**rounding**}
end for
$\Delta W_i = \sum_b \bar{g_b}$
output ΔW_i mod m for secure MPC

Algorithm 9 Server Training Procedure (detailed)

Input:

- the previous model state W_{t-1}
- the noise multiplier, σ
- the discretization granularity, γ
- m for modulus operation
- the client dataset sizes $||D_i||$ (assumed to be public). calculate the overall mini-batch size $B = p \cdot \sum_i^N ||D_i||$
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$
- the models updates $\overline{\Delta W} = ((\sum_i \Delta W_i' \bmod m) + (\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2) \bmod m)) \bmod m$ via secure MPC {this is equivalent to aggregating all scaled, flattened, and rounded gradient vectors: $= \left(\sum_b^B \bar{g}_b + (\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2))\right) \bmod m$}

1: Map $\overline{\Delta W}$ to $\overline{\Delta W}' \in [-m/2, m/2]^d \cap \mathbb{Z}^d$ $\{\overline{\Delta W}' \bmod m = \overline{\Delta W}\}$
2: $\Delta W = \gamma D_\xi H_d^T \overline{\Delta W}'$
3: output $W_t = W_{t-1} +$ optimizer$(\Delta W / B)$

$1 - x_i/\gamma$ *and* $\mathbb{P}[R_\gamma(x)_i = \gamma] = x_i/\gamma$. *In general, for* $x \in \mathbb{R}^d$, *we have* $R_\gamma(x) = \gamma\lfloor x/\gamma \rfloor + R_\gamma(x - \gamma\lfloor x/\gamma \rfloor)$; *here* $\gamma\lfloor x/\gamma \rfloor \in \gamma\mathbb{Z}^d$ *is the point* x *rounded down coordinate-wise to the grid.*

Definition 5 (Conditional Randomized rounding [20])**.** *Let* $\gamma > 0$ *and* $d \in \mathbb{N}$ *and* $G \subset \mathbb{R}^d$. *Define* $R_\gamma^G : \mathbb{R}^d \to \gamma\mathbb{Z}^d \cap G$ *to be* R_γ *conditioned on the output being in* G. *That is,* $\mathbb{P}[R_\gamma^G(x) = y] = \mathbb{P}[R_\gamma(x) = y]/\mathbb{P}[R_\gamma(x) \in G]$ *for all* $y \in \gamma\mathbb{Z}^d \cap G$, *where* R_γ *is as in Definition 4.*

Proposition 5 (Randomized Rounding + Centralized Discrete Gaussian)**.** *Leveraging Proposition 26 in [20], the utility for our centralized discrete Gaussian is as follows, with the difference from [20] which uses distributed discrete Gaussian noise highlighted in red:*

Let $\beta \in [0, 1)$, $\sigma^2 \geq \frac{1}{2}\gamma > 0$, *and* $c > 0$. *Let*

$$\Delta_2^2 := \min \left\{ \begin{array}{c} c^2 + \frac{1}{4}\gamma^2 d + \sqrt{2\log(1/\beta)} \cdot \gamma \cdot \left(c + \frac{1}{2}\gamma\sqrt{d}\right), \\ \left(c + \gamma\sqrt{d}\right)^2 \end{array} \right\},$$

$$G := \left\{ y \in \mathbb{R}^d : \|y\|_2^2 \leq \Delta_2^2 \right\},$$

$$\varepsilon := \frac{\Delta_2}{\sigma}.$$

Let R_γ^G *be as in Definition 5. Define a randomized algorithm* $A : (\mathbb{R}^d)^n \to \gamma\mathbb{Z}^d$ *by*

$$A(x) = \left[\sum_i^n R_\gamma^G \left(\min\left\{1, \frac{c}{\|x_i\|_2}\right\} \cdot x_i \right) \right] + \gamma \cdot Y, \tag{1}$$

where $Y \in \mathbb{Z}^d$ *are independent random vectors with each entry drawn independently from* $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2)$.

Then A satisfies $\frac{1}{2}\varepsilon^2$-concentrated differential privacy. The DP guarantee follows from the postprocessing property of DP and Proposition 5 and

Let $x_1, \cdots, x_n \in \mathbb{R}^d$ with $\|x_i\|_2 \le c$ for all $i \in [n]$. Then the following hold.

$$\left\| \mathbb{E}\left[A(x)\right] - \sum_i^n x_i \right\|_2 \le \frac{\beta \cdot \gamma \cdot \sqrt{d} \cdot n}{1-\beta}.$$

$$\mathbb{E}\left[\|A(x) - \mathbb{E}\left[A(x)\right]\|_2^2\right] \le \frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + d \cdot \sigma^2.$$

$$\mathbb{E}\left[\left\|A(x) - \sum_i^n x_i\right\|_2^2\right] \le \frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + \left(\frac{\beta}{1-\beta}\gamma\sqrt{d}n\right)^2 + d \cdot \sigma^2$$

$$\forall t \in \mathbb{R}^d \quad \mathbb{E}\left[\exp\left(\left\langle t, A(x) - \sum_i^n x_i \right\rangle\right)\right] \le \frac{\exp\left(\left(\frac{\gamma^2}{8} \cdot n + \frac{\sigma^2}{2}\right) \cdot \|t\|_2^2\right)}{(1-\beta)^n}.$$

Flattening [20] Since the inputs may be heavily concentrated on one coordinate. Hence, before modular clipping, the input vectors shall be flattened by multiplying them with a random unitary matrix or rotary $U \in \mathbb{R}^{d\times d}$, where $U^{-1} = U^T$. U^{-1} is multiplied to undo this operation at the end. Fix $x \in \mathbb{R}$ and $i \in [d]$. Then $\mathbb{E}\left[e^{t(Ux)_i}\right] \le e^{t^2\|x\|_2^2/2d}$, $\forall t \in \mathbb{R}$ and $\forall i \in [d]$. However, simply using a unitary or rotary matrix here is not ideal due to several reasons. We followed the same flattening procedure as proposed by [20]: let $H \in \{-\sqrt{\rho/d}, +\sqrt{\rho/d}\}$ be a Walsha-Hadamard matrix satisfying $H^T H = I$, let $\xi \in \{-1, +1\}^d$ be the public uniformly random sign vector and $D_\xi \in \{-1, 0, +1\}^{d\times d}$ is a diagonal matrix with ξ on the diagonal. Fix $x \in \mathbb{R}$ and $i \in [d]$. Flattening is the operation $y = HD_\xi x$, which has the desired utility guarantee: let $Y = (HD_\xi x)_i \in \mathbb{R}$. Then $\mathbb{E}\left[e^{tY}\right] \le e^{t^2\|x\|_2^2\rho/2d}$, $\forall t \in \mathbb{R}$. Optimality is attained when $\rho = 1$.

Definition 6 (Modular clipping [20])**.** *For $a < b$, define $M_{[a,b]} : \mathbb{R} \to [a, b]$ by $M(x) = x + (b-a)\cdot n$ where $n \in \mathbb{Z}$ is chosen so that $x + (b-a)\cdot n \in [a, b]$. (Ties are broken arbitrarily.) We also define $M_{[a,b]}(x) = (M_{[a,b]}(x_1), M_{[a,b]}(x_2), \cdots, M_{[a,b]}(x_d)) \in [a, b]^d$ for $x \in \mathbb{R}^d$. It has the property that* $\forall a < b \quad \forall x, y \in \mathbb{R} \qquad M_{[a,b]}(x + y) = M_{[a,b]}(M_{[a,b]}(x) + M_{[a,b]}(y))$.

Theorem 6 (Randomized rounding, flattening, modular clipping, and centralized discrete Gaussian)**.** *Leveraging Theorem 36 in [20], the utility for our centralized discrete Gaussian is as follows, with the difference from [20] which uses distributed discrete Gaussian noise highlighted in red: Let $\beta \in [0, 1)$, $\sigma^2 \ge \frac{1}{2}\gamma > 0$, and $c > 0$. Let $n, d \in \mathbb{N}$ and $\rho \ge 1$. Let $U \in \mathbb{R}^{d\times d}$ be a random unitary matrix such that*

$$\forall x \in \mathbb{R}^d \quad \forall i \in [d] \quad \forall t \in \mathbb{R} \qquad \mathbb{E}\left[\exp(t(Ux)_i)\right] \le \exp(t^2\rho\|x\|_2^2/2d).$$

Let

$$\Delta_2^2 := \min\left\{ \begin{array}{c} c^2 + \frac{1}{4}\gamma^2 d + \sqrt{2\log(1/\beta)}\cdot\gamma\cdot\left(c + \frac{1}{2}\gamma\sqrt{d}\right), \\ \left(c + \gamma\sqrt{d}\right)^2 \end{array} \right\},$$

$$G := \left\{y \in \mathbb{R}^d : \|y\|_2^2 \le \Delta_2^2\right\},$$

$$\varepsilon := \frac{\Delta_2}{\sigma}.$$

Let R_γ^G be as in Definition 5. Let $r > 0$ and let $M_{[-r,r]}$ be as in Definition 6. Let $A(x)$ be as defined in Equation 1 We have

$$\left\| \mathbb{E}\left[A(Ux)\right] - \sum_i^n x_i \right\|_2 \leq \frac{\beta \cdot \gamma \cdot \sqrt{d} \cdot n}{1-\beta}.$$

$$\mathbb{E}\left[\left\|A(Ux) - \mathbb{E}\left[A(Ux)\right]\right\|_2^2\right] \leq \frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + d \cdot \sigma^2.$$

$$\mathbb{E}\left[\left\|A(Ux) - U\sum_i^n x_i\right\|_2^2\right] \leq \frac{\gamma^2 \cdot d \cdot n}{4(1-\beta)} + \left(\frac{\beta}{1-\beta}\gamma\sqrt{d}n\right)^2 + d \cdot \sigma^2$$

$$\mathbb{E}\left[\exp\left(t \cdot \left(A(Ux) - U\sum_i^n x_i\right)_j\right)\right] \leq \frac{\exp\left(\left(\frac{\gamma^2}{8}\cdot n + \frac{\sigma^2}{2}\right)\cdot t^2\right)}{(1-\beta)^n}, \quad \forall t \in \mathbb{R}\ \forall j \in [d]$$

$$\mathbb{E}\left[\exp\left(t \cdot \left(A(Ux)\right)_j\right)\right] \leq \exp\left(\frac{t^2\rho}{2d}\left\|\sum_i^n x_i\right\|_2^2\right) \cdot \frac{\exp\left(\left(\frac{\gamma^2}{8}\cdot n + \frac{\sigma^2}{2}\right)\cdot t^2\right)}{(1-\beta)^n} \quad \forall t \in \mathbb{R}\ \forall j \in [d].$$

Define a randomized algorithm $\tilde{A} : (\mathbb{R}^d)^n \to \gamma\mathbb{Z}^d$ by

$$\tilde{A}(x) = U^T M_{[-r,r]}\left(\left(\sum_i^n R_\gamma^G\left(\min\left\{1, \frac{c}{\|x_i\|_2}\right\} \cdot Ux_i\right)\right) + \gamma \cdot Y\right), \tag{2}$$

where $Y \in \mathbb{Z}^d$ are independent random vectors with each entry drawn independently from $\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2)$.

Then $\tilde{A}$ satisfies $\frac{1}{2}\varepsilon^2$-concentrated differential privacy.

Let $x_1, \cdots, x_n \in \mathbb{R}^d$ with $\|x_i\|_2 \leq c$ for all $i \in [n]$. Let

$$\hat{\sigma}^2(x) := \frac{\rho}{d}\left\|\sum_i^n x_i\right\|_2^2 + \left(\frac{\gamma^2}{4} + \sigma^2\right)\cdot n \leq \frac{\rho}{d}c^2n^2 + \left(\frac{\gamma^2}{4}\cdot n + \sigma^2\right) \tag{3}$$

If $\hat{\sigma}^2(x) \leq r^2$, then

$$\mathbb{E}\left[\left\|\tilde{A}(x) - \sum_i^n x_i\right\|_2^2\right] \tag{4}$$

$$\leq \frac{d \cdot n}{1-\beta} \cdot \left(\frac{2\sqrt{2} \cdot r \cdot e^{-r^2/4\hat{\sigma}^2(x)}}{\sqrt{n \cdot (1-\beta)^{n-1}}} + \sqrt{\frac{\gamma^2}{4} + \frac{\beta^2\gamma^2 n}{1-\beta} + \frac{(1-\beta)\sigma^2}{n}}\right)^2. \tag{5}$$

Algorithm 10 Client Training Procedure

Input:

- the current model state W
- the loss function L
- the clipping norm, c
- the noise multiplier, σ
- the discretization granularity, γ
- the bias β
- the private training data D_i for client i,
- m for modulus operation
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$

1: $g_i \leftarrow$ local model update on local data D_i
2: $g_i' \leftarrow \frac{1}{\gamma}(g_i \cdot \min\{1, \frac{c}{||g_i||_2}\})$ {clip and scale the gradient}
3: $g_i'' \leftarrow H_d D_\xi g_i'$ where $H \in \{-1/\sqrt{d}, +1/\sqrt{d}\}$ is a Walsha-Hadamard matrix satisfying $H^T H = I$ and $D_\xi \in \{-1, 0, +1\}^{d \times d}$ is a diagonal matrix with ξ on the diagonal {flatten the gradient}
4: repeat $\tilde{g}_i = \texttt{randomized round}(g_i'')$ until $||\tilde{g}_i||_2 \leq \min\left\{c/\gamma + \sqrt{d}, \sqrt{c^2/\gamma^2 + \frac{1}{4}d + \sqrt{2\log 1/\beta} \cdot (c/\gamma + \frac{1}{2}\sqrt{d})}\right\}$
5: $\Delta W_i = \sum_b \bar{g}_i$
6: output $\Delta W_i \bmod m$ for secure MPC

D For per-client privacy guarantee

To leverage the centralized noise sampling method for client-level privacy guarantee, one can follow Algorithms 10 and 11, which is equivalent replacing the distributed noise with centralized noise of the main algorithm in [20]. The utility analysis for the Algorithms 10 and 11 is the same as described in Appendix C, where n here represents the number of clients that aggregate the client updates, instead of the aggregate mini-batch sizes.

Algorithm 11 Server Training Procedure

Input:

- the previous model state W_{t-1}
- the number of clients in this training run n_clients
- the noise multiplier, σ
- the discretization granularity, γ
- m for modulus operation
- Public uniformly random sign vector $\xi \in \{-1, +1\}^d$
- the models updates $\overline{\Delta W} = ((\sum_i \Delta W_i' \bmod m) + (\mathcal{N}_{\mathbb{Z}}(0, \sigma^2/\gamma^2) \bmod m)) \bmod m$ via secure MPC

1: Map $\overline{\Delta W}$ to $\overline{\Delta W}' \in [-m/2, m/2]^d \cap \mathbb{Z}^d$ $\{\overline{\Delta W}' \bmod m = \overline{\Delta W}\}$
2: $\Delta W = \gamma D_\xi H_d^T \overline{\Delta W}'$
3: output $W_t = W_{t-1} +$ optimizer$(\Delta W / n_\text{clients})$

E Additional Experimental Setup

In this work, we closely followed the algorithms and notations in [20] to scale, flatten, round, and modular clip the gradients update produced by each participants. For each vector $x \in \mathcal{R}^d$, $d \in \mathbb{N}$ to be discretized, let $\gamma > 0$ be the granularity for discretization. We use the default values as in [20]: the bias $\beta = e^{-1/2}$, range for modular clipping $m = 2^{16}$ (i.e., 16 bits); for a fixed number of bits (n_bits), we use the heuristics to choose the discretization granularity γ by making sure the range m includes k standard deviations of $(\sum_b^B \bar{g}_b) + Y$, i.e., that $2k\hat{\sigma} \leq 2^{n\_bits} \cdot \gamma$ ($\bar{g}_b, \gamma, B$ and Y are as defined in Algorithms 6 and 7), where $\hat{\sigma}^2 = \frac{1}{d}c^2B^2 + \left(\frac{\gamma^2}{4} \cdot B + \sigma^2\right)$. $k = 4$ is used in all our experiments. This is similar to the one described in [20] with the difference highlighted in red. We use similar hyperparameters for FL with data point-level privacy protection (Section 5.2) as prior work on DP fine-tuning soft prompts [7], as summarized in Table 4.

F Additional Experiments

F.1 Latency Dependence for Prior Work

We benchmark the cost of prior work for four parties for different latency settings and a bandwidth up to 1 Gbps as shown in Figure 9.

F.2 DP-FL additional experiments

Additional experimental results on QNLI, QQP, and MNLI are shown in Figures 10, 11, and 12 respectively. The experiments are conducted for five runs in each case. The final accuracy scores

Table 4: **Hyperparameters for the experiments.** All experiments use 10 clients. The datasets are equally splitted to simulate the local data of the clients. RoBERTa-base model is used for all experiments. The soft prompts and the classifier layers of the models are fine-tuned while all other parameters are frozen during the training. BS is the aggregate batch size (i.e., $B = \sum_i B_i$ from Algorithm 7). LR is the learning rate. ϵ is the privacy budget. Grad is the maximum gradient norm c as defined in Algorithm 6. p-len is the length of the soft prompts.

data	**model**	**BS**	**LR**	ϵ	**Grad**	**Epochs**	**p-len**
sst2	base	900	0.05	8.0	0.01	21	9
QNLI	base	1050	0.005	8.0	0.05	100	10
QQP	base	1050	0.05	8.0	0.1	10	7
MNLI	base	1050	0.005	8.0	0.05	60	10

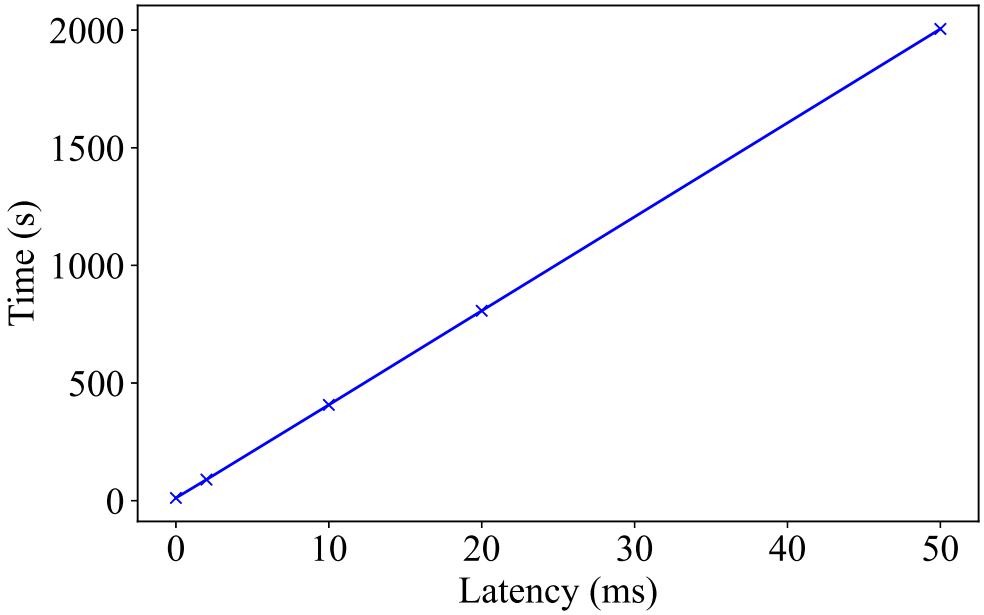

Figure 9: **Latency vs Time for [34].** Performance of prior work for standard deviation 967 with 4 parties and a bandwidth of up to 1 Gbps.

(mean ± std) are shown in Table 5.

Table 5: **Summary of evaluation accuracy values (%).** All experiments are run five times. We report the mean ± standard deviation.

data	Baseline	Ours	5 colluding	9 colluding
sst2	90.37 ± 0.56	89.68 ± 0.85	88.51 ± 1.18	83.44 ± 0.50
QNLI	81.95 ± 0.75	82.37 ± 0.37	81.86 ± 0.37	79.33 ± 0.52
QQP	76.34 ± 0.50	77.31 ± 0.84	75.34 ± 0.89	71.05 ± 3.17
MNLI	73.63 ± 0.71	73.60 ± 0.49	71.96 ± 0.68	55.92 ± 4.25

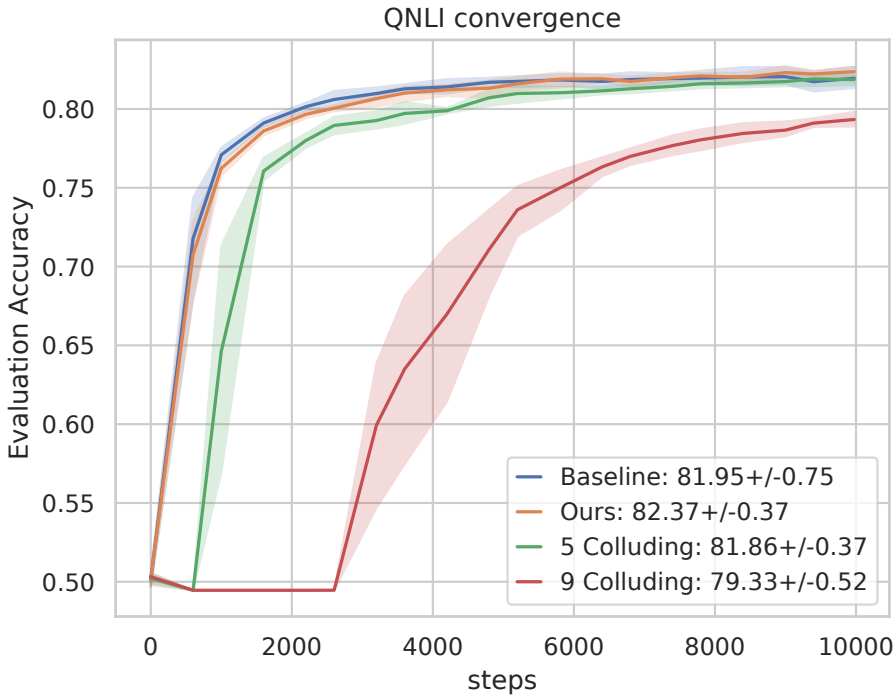

Figure 10: **The convergence plots on QNLI dataset.** This is a reproduction of Figure 8 using QNLI dataset with five training runs for each of the four cases.

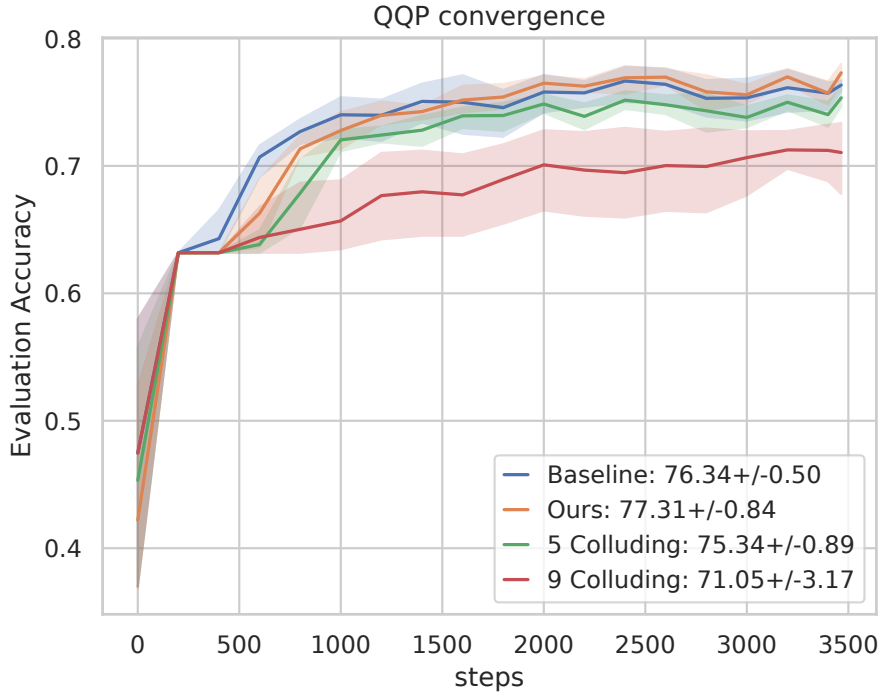

Figure 11: **The convergence plots on QQP dataset.** This is a reproduction of Figure 8 using QQP dataset with five training runs for each of the four cases.

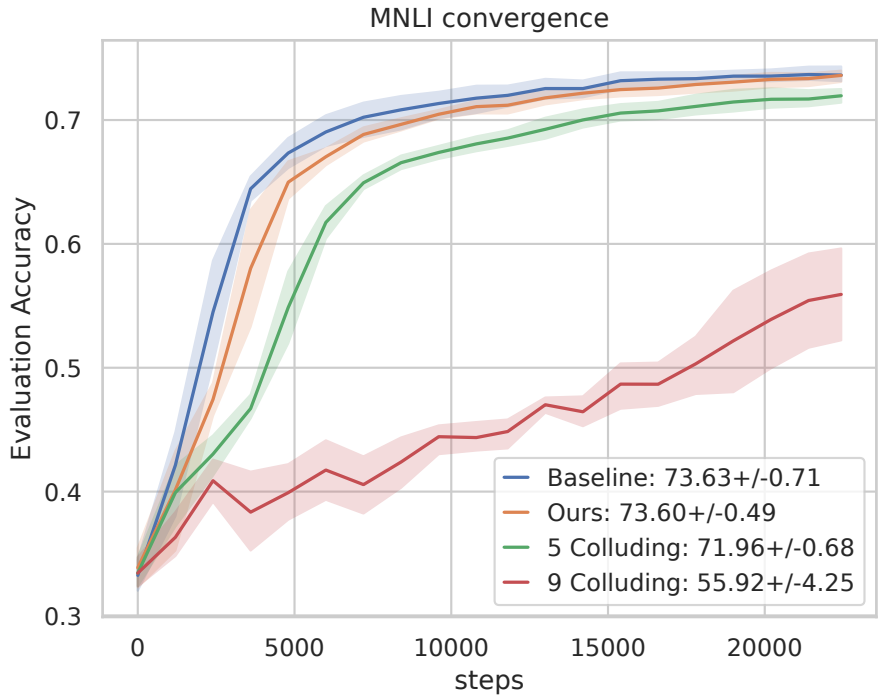

Figure 12: **The convergence plots on MNLI dataset.** This is a reproduction of Figure 8 using MNLI dataset with five training runs for each of the four cases.