On Evolution of Ransomware & Malware Analysis in Cryptography

Deepak *

Abstract

***Abstract*—In collaborative learning (CL), multiple parties jointly train a machine learning model on their private datasets. However, data can not be shared directly due to privacy concerns. To ensure *input confidentiality*, cryptographic techniques, e.g., multi-party computation (MPC), enable training on encrypted data. Yet, even securely trained models are vulnerable to inference attacks aiming to extract memorized data from model outputs. To ensure *output privacy* and mitigate inference attacks, differential privacy (DP) injects calibrated noise during training. While cryptography and DP offer complementary guarantees, combining them efficiently for cryptographic and differentially private CL (CPCL) is challenging. Cryptography incurs performance overheads, while DP degrades accuracy, creating a privacy-accuracy-performance trade-off that needs careful design considerations. This work systematizes the CPCL landscape. We introduce a unified framework that generalizes common phases across CPCL paradigms, and identify secure noise sampling as the foundational phase to achieve CPCL. We analyze trade-offs of different secure noise sampling techniques, noise types, and DP mechanisms discussing their implementation challenges and evaluating their accuracy and cryptographic overhead across CPCL paradigms. Additionally, we implement identified secure noise sampling options in MPC and evaluate their computation and communication costs in WAN and LAN. Finally, we propose future research directions based on identified key observations, gaps and possible enhancements in the literature.**

***Index Terms*—Differential privacy, cryptography, collaborative machine learning**

I. Introduction

Strict privacy laws, e.g., GDPR [1], along with concerns over data misuse and breaches hinder direct data sharing among multiple parties to collaboratively train machine learning models. Cryptographic techniques, e.g., *multi party computation* (MPC) [2] and *homomorphic encryption* (HE) [3], enable joint training on private datasets by encrypting data during training. However, they do not prevent models from leaking information on training data during inference. Thus, models remain vulnerable to inference attacks, e.g, membership inference, which can reveal if a specific sample was in the training data [4]. To mitigate inference attacks, *differential privacy* (DP) [5] bounds the information leakage by injecting carefully calibrated noise during training. In the central DP model (CDP), users send raw data to a trusted third party (TTP) to add noise to the computation output. To avoid sharing raw data, in the local model (LDP), each user adds noise to its data. However, LDP yields lower accuracy than CDP. For example, Google's LDP telemetry system [6] failed to detect a common signal among 1 million users, despite billions of user reports. Enhancing cryptographic collaborative learning with DP to realize *cryptographic and differentially private collaborative learning* (CPCL) provides: (I) *input confidentiality* via cryptography, (II) *output privacy* via DP, and (III) *high accuracy* via secure sampling of CDP noise without a TTP. CPCL is an emerging topic with growing interest from academia [7]–[11] and industry [12]–[14]. For example, Google's large-scale deployment [13] enables next-word predictions for Gboard keyboards by aggregating masked noisy information from clients to satisfy DP. Apple uses DP and secure aggregation to learn popular scenes photographed by iOS users to create personalized Memories [15]. While various SoKs [16]–[18] cover cryptographic CL, the integration of DP in cryptographic CL lacks a comprehensive systematization of key techniques, design considerations, and trade-offs. This work bridges this gap by introducing a comprehensive framework for CPCL, analyzing secure noise sampling techniques, and evaluating performance-accuracy trade-offs across learning paradigms. From our systematization, we identify two main learning paradigms: *federated learning* (FL) and *outsourced learning* (OL). In FL, data-holding clients iteratively encrypt and send local model updates to servers aggregating them into a global update (Sec. IV-C). In contrast, OL clients send their encrypted data to servers to train on global encrypted data (Sec. IV-D). While cryptography and DP offer complementary guarantees, their integration is challenging since both introduce performance overhead and accuracy trade-offs:

Cryptography performance overhead: cryptographic techniques incur high communication/computation costs, e.g., computation-intensive HE or communication-intensive MPC. Overhead also depends on the learning paradigm: Sec. VI shows that OL can be $10^3\times$ slower than plaintext, while FL is $10\times$ faster than OL but leaks intermediate model updates.

Cryptography performance-accuracy trade-off: cryptographic techniques rely on fixed-point arithmetic trading accuracy for efficiency [2], and introducing numerical errors. OL approximates non-linear operations, e.g., Softmax, affect-

*Work done while he was at SAP SE.

This work has been accepted for publication at the IEEE Conference on Secure and Trustworthy Machine Learning (SaTML 2026). The final version will be available on IEEE Xplore.

ing accuracy (Sec. VI), while FL quantizes local updates to reduce communication, introducing approximation errors.

DP performance overhead: DP requires noise sampling and per-example gradient clipping. Naive clipping can slow training by 200× (for a 3-layer NN, Sec. VI), while secure noise sampling further increases overhead (Sec. V).

DP privacy-accuracy trade-off: DP noise degrades accuracy and must be calibrated to the threat model. Accuracy depends on *where* noise is injected (Alg. 1), *who* samples it (Alg. 2), *how* it is sampled (Sec. V), and *which* distribution (Tab. II).

Contributions. We provide a thorough and structured analysis of solutions combining cryptography and DP for CL. While related works (Sec. VII) focus on either cryptographic CL (FL [16], [17] or OL [18]) or DP in isolation [19], [20], we systematize their holistic integration. Our contributions are:

- We introduce a comprehensive framework to model and generalize CPCL solutions; we identify common phases across learning paradigms, distinguishing steps on local versus (securely joint) global data and possible design choices (Fig. 1, Alg. 2). This enables us to highlight emerging trends and gaps in the CPCL literature as well as possible enhancements to existing solutions (Tab. I).
- We focus on noise sampling as foundational phase in CPCL to integrate DP guarantees into cryptographic CL. We provide an in-depth analysis of secure techniques for distributed noise generation (Sec. V), and detail noise distributions (Tab. II) as well as sampling approaches (Alg. 2).
- We implement noise sampling in MPC, evaluate computation and communication costs for semi-honest and malicious schemes in LAN and WAN (Sec. V-C).[1]
- We evaluate and compare privacy-accuracy-performance trade-offs across CPCL paradigms and noise sampling techniques (Sec. VI).[1] Our analysis offers insights and guidelines to enhance cryptographic CL with DP (Sec. VI-A).
- Throughout our work, we distill key observations (**O#**) from which we derive research directions (**D#**) (Sec. VIII).

II. Scope & Methodology

Next, we outline our scope and methodology.

Scope. This work systematizes the landscape of CPCL, focusing on existing and possible techniques and trade-offs to combine DP and cryptography for CL. We include only works co-designing solutions to integrate DP and cryptography in CL. We do not systematize works relying solely on cryptography, e.g., secure aggregation [4], [21], as they do not provide output privacy, and those using only DP, due to strong trust assumptions (CDP) or reduced accuracy (LDP). Rather, we leverage our proposed framework (Sec. IV-B) to discuss design choices on how those can be augmented to achieve CPCL (Sec. VI-A). We focus on horizontally partitioned datasets, where clients hold disjoint records with the same features as it is the most common setting in CL and aligns with related SoKs [16]–[18]. We do not detail vertical partitioning [22], [23], and split learning [24]–[26] since they introduce challenges beyond our scope, e.g., entity alignment, hot to split the model, but we map them to our framework in App. F.

Methodology. We started by performing a systematic search across top-ranked venues for security, cryptography, and ML, based on established rankings [27]–[29]. We performed an extensive search on Google Scholar and BASE with specific keywords referencing our scope, e.g., differential privacy, cryptography, collaborative, training, from 2018. We detail search keywords and venues in App. A. Our systematic search returned 650 papers. After examining the abstracts we identified 61 potentially in-scope works. After a thorough analysis, we selected 11 works integrating cryptography and DP in CL. For comprehensive coverage, we expanded our search by applying the above criteria to works citing or being cited by selected works. We included 11 further works, for a total of 22 relevant works categorized in Tab. I.

III. Preliminaries

Before introducing preliminaries for DP, cryptography, and privacy attacks, we recall *Gradient Descent* (GD) in ML training. GD is an optimization technique to find a set of optimal parameters θ^* to minimize a loss $L(\theta)$ of the model over a training dataset. For each training step k, GD updates the parameters in the opposite direction of the gradient of the loss: $\theta^{(k+1)} = \theta^{(k)} - \eta \nabla L(\theta^{(k)})$. Here, η is the learning rate. For efficiency on large datasets, *stochastic gradient descent* (SGD) approximates GD by using only a random subset of the training data, or *batch* of size B, at each step [30].

A. Differential Privacy

DP is a privacy definition that guarantees that the inclusion or exclusion of a record in an analysis does not significantly impact the result. Formally, a randomized mechanism $\mathcal{M}$ satisfies (ϵ, δ)-DP if for any neighboring dataset D_1, D_2 (differing in at most one record), and for any subset $\mathcal{S} \subseteq \mathrm{Range}(\mathcal{M})$:

$$\Pr[\mathcal{M}(D_1) \in \mathcal{S}] \leq \exp(\epsilon) \times \Pr[\mathcal{M}(D_2) \in \mathcal{S}] + \delta. \quad (1)$$

Here, $\epsilon > 0$ is the *privacy budget* and bounds output difference over $\mathcal{M}$ on D_1, D_2. Smaller ϵ indicates stronger protection. Parameter δ models the probability of violating this privacy guarantee. For $\delta = 0$, we get *pure* DP (ϵ-DP), whereas for $\delta > 0$ we get *approximated* DP. Typically, for ML $\delta \ll \frac{1}{N}$, where N is the dataset size [19]. DP in Eq. (1) ensures record-level protection, as D_1, D_2 differ by one record [31]. User-level protection must account for users contributing multiple records, i.e., D_1, D_2 differ by an entire user's data (Sec. VI).

DP Mechanisms. To satisfy DP, a mechanism $\mathcal{M}$ can add noise to a function output. Formally, $\mathcal{M}(D, f(\cdot)) = f(D) + \psi$, where $f : \mathcal{D}^n \to \mathbb{R}$ is a function applied over a dataset D, and ψ is the noise sampled from a suitable distribution (Tab. II). Typically, the noise is calibrated on the f's l_p-sensitivity: $\Delta_p = \max_{D_1, D_2} ||f(D_1) - f(D_2)||_p$, where D_1, D_2 are neighboring datasets. In CL with multiple data owners different variants of DP can be used. In *central* DP (CDP), data owners send their data to a Trusted Third Party (TTP) that applies $\mathcal{M}$ on the data. To avoid a TTP, in *local* DP (LDP) [32], each

[1] Code available at: https://github.com/SAP/sok-cpcl

Input: Training data $D = \{x_i, y_i\}_{i=1}^n$, and initial parameters $\theta^{(0)}$
Output: Model parameters θ
$x \leftarrow \mathsf{PerturbInput}(x)$
foreach *training step* k **do**
  $L(\theta^{(k)}, D) \leftarrow \mathsf{PerturbLoss}(\frac{1}{n}\sum_{i=1}^n \ell(\theta_i^{(k)}, x_i, y_i))$
  $g \leftarrow \mathsf{PerturbGradient}(\nabla_\theta L(\theta^{(i)}, D))$
  $\theta^{(k+1)} \leftarrow \theta^{(k)} - \eta \cdot g$
return $\mathsf{PerturbOutput}(\theta)$

Algorithm 1: Training perturb options (based on [38])

data owner applies the perturbation independently (App. C-A). LDP is a stricter guarantee than CDP, as it requires $\mathcal{M}$ to give bounded indistinguishable output between any possible pair of data points x_1, x_2 from D [19]. However, this stronger privacy guarantee incurs an accuracy cost. For n-party count queries, LDP yields $O(\sqrt{n})$ error, while CDP's error is $O(1)$ [33]. The *shuffle model* [6] improves over LDP by using a TTP to shuffle messages from data owners and break user-message correlation with $O(\log n)$ error [33]. While we exclude this model as it has weaker accuracy than CDP, we include works that employ shuffling to achieve CDP guarantees [11], [34].

DP Properties for ML. DP is immune to *post-processing*, i.e., an attacker can not weaken the privacy of a DP output. Furthermore, composition of multiple DP mechanisms remains DP, crucial for tracking privacy budget across ML training iterations. *Basic composition* defines the worst (ϵ, δ)-bound for n (ϵ_i, δ_i)-DP mechanisms applied on a dataset, i.e., equivalent to applying a $(\sum_{i=1}^n \epsilon_i, \sum_{i=1}^n \delta_i)$-DP mechanism. However, basic composition is overly conservative, especially for ML. To improve composition relaxed DP definitions like zero-Concentrated DP (zCDP) [35] and Rényi DP (RDP) [36] provide tighter bounds using tools like *moments accountant* [37].

DP Options in ML. As Alg. 1 shows, DP noise can be applied at different training stages. PerturbInput randomizes input data [39], PerturbLoss perturbs the loss [40], PerturbGradient applies DP noise to gradients [5], and PerturbOutput perturbs model parameters after training [41]. Notably, PerturbInput can obscure patterns, hindering learning. PerturbLoss assumes a strongly convex loss function which is typically not the case in neural networks (NN) [38]. PerturbOutput is limited to simple models (e.g., linear regression) since NNs have complex dependencies between data and weights, making sensitivity analysis infeasible [42]. Another approach, PerturbLabel [43] (App. G) adds noise to predicted labels during inference. However, training with DP labels only ensures label-DP [44]. Thus, PerturbGradient is the most practical approach suitable for any gradient-based optimization method like SGD [19], as discussed next.

DP-SGD. To make SGD differentially private (DP-SGD) [37], [45], PerturbGradient modifies SGD as follows:

$$\begin{aligned} \nabla L'(\theta^{(k)}) &= \nabla L(\theta^{(k)}) \cdot \min(1, K/||\nabla L(\theta^{(k)})||_2), \\ \theta^{(k+1)} &= \theta^{(k)} - \eta(\nabla L'(\theta^{(k)}) + \psi). \end{aligned} \tag{2}$$

Here, ψ is a noise sample drawn from a suitable distribution (Tab. II), and K is the clipping parameter. The noise magnitude depends on the sensitivity of the gradient computation, i.e., the norm. Since gradient norm can be unbounded, each gradient l_2 norm is clipped to K [37]. Typically, clipping requires computing B per-example gradients, instead of one gradient over the batch loss, increasing computational overhead (Sec. VI). To mitigate noise impact, *gradient accumulation* averages gradients over B samples, thereby averaging also the noise. Privacy amplification by *subsampling* enhances DP-SGD's privacy-accuracy trade-off by sampling a subset of data per iteration, leveraging the uncertainty of sample inclusion (App. D).

B. Cryptography

First, we introduce security models, party roles, and notations, followed by an overview of cryptographic techniques.

Security Models, Parties & Notation. Cryptographic security models define adversarial capabilities. In the *semi-honest* model, a passive attacker follows the protocol but tries to extract private information. In contrast, in the *malicious* model, an active attacker can deviate from the protocol, e.g., manipulate inputs to alter outputs. In both, up to t *colluding* parties can jointly try to infer others' private input. We distinguish three roles: *input parties* own and provide data; *computing parties* execute cryptographic protocols; and *output parties* $\mathcal{O}$ receive results. We refer to computing parties as *servers* $\mathcal{S}$, while *clients* $\mathcal{C}$ are input and output parties. In specific settings, $\mathcal{C}$ also act as computing parties, and $\mathcal{S}$ as output parties. Unless noted otherwise, we consider n semi-honest clients, m semi-honest servers, and up to t colluding parties ($\mathcal{C}$, $\mathcal{S}$). In some settings, we consider a semi-trusted server S' which does not have access to computation outputs, and can, e.g., sample DP noise in cleartext (Sec. V-A). We denote encrypted values x as $[x]$, and highlight computations on joint, encrypted data.

Masking. Pair-wise masking [46] enables secure aggregation (SecAgg), using additions modulo r, with a single server S ($|\mathcal{S}| = 1$). Each pair of clients (C_i, C_j) shares a mask b_{ij} via, e.g., Diffie-Hellman key exchange [47]. C_i adds b_{ij} to its data, i.e., $[x_i] = (x_i + b_{ij}) \bmod r$, while C_j adds $-b_{ij}$. When S sums those values, the masks cancel out, and reveal only the aggregated output, i.e., $[x_i] + [x_j] = (x_i + b_{ij}) + (x_j - b_{ij}) = x_i + x_j$. An alternative solution is based on *learning with error* (LWE) [48]. In LWE, an error vector e breaks the linearity of a set of equations $b_i = As_i + e_i$. Each client C_i holds b_i as a mask, s_i as a secret key, and publicly shares the matrix A. After aggregation, the clients reconstruct $\sum_i s_i$ to remove the mask $\sum_i b_i = \sum_i (As_i + e_i)$, where $e = \sum_i e_i$ can additionally provide DP [49] as we illustrate in Sec. V-B.

Homomorphic Encryption (HE). HE [3] enables computing on encrypted data with a single server S ($|\mathcal{S}| = 1$). All clients share a public key pk, and each client C_i encrypts its data, i.e., $[x_i] = \mathsf{Enc}_{pk}(x_i)$, and sends $[x_i]$ to S which computes on the encrypted data. Clients can decrypt the result with the corresponding private key sk, i.e., $\sum_i x_i = \mathsf{Dec}_{sk}(\sum_i [x_i])$. Variants include, *additive* HE (AHE) [50] for encrypted sums; *fully* HE (FHE) [51] for both multiplication and additions (to evaluate arbitrary functions); and threshold HE [52], [53], where sk is secret shared among the clients, and decryption requires at least $(t+1) \leq n$ clients.

Multi-Party Computation (MPC). MPC allows multiple parties to jointly compute a function while keeping their inputs private with multiple servers ($|\mathcal{S}| > 1$). Typically, MPC is split

in a slow, data-independent *offline phase* for pre-computation, and a fast, data-dependent *online phase* consuming offline material [2]. A common paradigm for MPC is *threshold secret sharing* ((t, m)-SS), where each client C_i splits its data x_i into m parts, called *shares*, i.e., $[x_i] = \mathsf{Shr}(x_i)$, distributed to m servers. The servers compute on shares locally (e.g., addition) or interactively (e.g., multiplication). At least $t+1$ servers are required to *reconstruct* results, i.e., $\mathsf{Rec}([x_1], ..., [x_m])$. The threshold t ensures the (t, m)-SS scheme is secure against up to t colluding servers. Another paradigm that can securely evaluate arbitrary functions is *garbled circuits* (GC, App. E-A) [54], which is more suited than SS for Boolean operations.

Cryptography and DP. Combining cryptography and DP requires accounting for computational security of cryptographic schemes in the DP guarantee. Computational DP [31] (formalized in App. C-B) adapts the DP definition (Def. 1) to consider a bounded polynomial-time adversary by adding the negligible failure probability of cryptographic schemes to DP's δ. Hence, only approximated DP ($\delta > 0$) is achievable with cryptography. Additionally, sampling DP noise via cryptographic protocols can fail due to finite-precision arithmetic. Keller et al. [55] absorb this failure probability, e.g., due to overflows or precision mismatch, into DP's δ. Although increasing δ theoretically affects privacy accounting, the impact of typical security parameters (e.g., 2^{-128}) is negligible.

C. Privacy Attacks in CL

Next, we discuss attacks in the semi-honest setting. We detail further security models, attacks, and mitigations in App. P

Membership Inference Attacks (MIA). MIA aim to infer whether a specific record (or user) was part of the training data by exploiting differences in model behavior, e.g., confidence scores, between training and non-training data, often due to overfitting [4]. DP training mitigates MIA by bounding the influence of any data point on the trained model [56], [57].

Gradient Inversion Attacks (GIA). GIA aim to reconstruct training data from gradients computed during training. In FL, an adversary corrupting a client or server, has access to aggregated gradients and can reconstruct training samples by optimizing a loss function to match the observed gradients [58]. GIA mitigations rely on cryptographic techniques that ensures gradient secrecy against $\mathcal{S}$, e.g., HE with a single server [59] or SS with multiple servers (see Sec. IV-C).

IV. Encrypted and DP Collaborative Learning

Next, we systematize the CPCL landscape in a top-down approach. We overview building blocks in Sec. IV-A, identify common phases in Sec. IV-B, and analyze trade-offs for FL and OL in Sec. IV-C, IV-D, respectively. We highlight forward references to detailed discussions with ▶ within the overview.

A. Systematization

Next, we introduce core building blocks of CPCL, via columns from Tab. I, discussing them from left to right. Tab. I categorizes CPCL works, highlights trends, identifies unexplored combinations of techniques, i.e., gaps (−), and potential enhancements using existing techniques (⊛). For convenience, we provide a summary of notation in Tab. VI (App. B).

Learning Paradigms. We identify two main *learning paradigms* in CPCL: federated learning (FL) and outsourced learning (OL). In FL, clients train local models on their private data and send encrypted updates to a server for aggregation. In OL, clients outsource training to servers that compute on encrypted data. Tab. I shows that FL is the main paradigm in CPCL, likely due to its efficiency, i.e., only requiring additions on encrypted data. We analyze FL and OL trade-offs in ▶ Sec. IV-C, IV-D, and evaluate them in ▶ Sec. VI.

Noise Generation. For *noise generation* we distinguish five aspects. We identify two noise *types*: CNoise, a single CDP noise term sampled globally, and PNoise, a partial non-DP noise term sampled locally by each client or server. While a single PNoise does not satisfy DP, the aggregation of multiple PNoise terms satisfies CDP. We further distinguish three *sampling techniques*: centralized CNoise sampling via a semi-trusted server, local PNoise sampling by individual clients/servers, and distributed CNoise sampling via MPC across multiple servers. Notably, while PNoise is sampled independently and added locally to encrypted local updates, CNoise is sampled once and applied to already-aggregated data. The two main perturbation options are: PerturbGradient and PerturbOutput. Different *mechanisms* are used by systematized works to sample noise from suitable distributions. We also distinguish the *sampler*, which can be either clients or servers. Among noise types, PNoise is the most common, since it is the most efficient requiring only local sampling. Furthermore, PNoise has been implemented with a variety of mechanisms, while CNoise is limited to Laplace and Gaussian. None of the analyzed works adopt distributed CNoise sampling with PerturbGradient, likely due to performance overhead. Notably, centralized CNoise sampling is used by only one FL work [34], as it requires a semi-trusted server. In the rest of this work, we focus mainly on PerturbGradient, the most flexible and common option, but also highlight differences with PerturbOutput. We detail noise mechanism in ▶ Tab. II, Sec. V, analyze in-depth noise generation techniques in ▶ Sec. V-A, V-B, V-C, and evaluate privacy-accuracy-performance trade-offs in ▶ Tab. IV-V, Sec. VI.

Security Properties. We distinguish *gradient secrecy* which hides aggregated DP gradients preventing gradient inversion attacks, and *model secrecy* which protects the parameters of the trained model, preserving IP. OL inherently guarantees both, while FL requires cryptographic enhancements (⊛). Notably, FL with PerturbOutput guarantees gradient secrecy by design, as only noisy model parameters are revealed. We also identify *oblivious noise* when output parties do not know the sampled noise, preventing its removal and preserving privacy guarantees while improving accuracy. While distributed CNoise guarantees noise obliviousness by default, PNoise requires cryptographic enhancements (⊛) to prevent corrupted parties from removing their noise contributions. We discuss security properties for FL and OL in ▶ Sec. IV-C, IV-D;

TABLE I: Categorization of CPCL Papers by Learning Paradigm, DP Noise and security properties. Perturbations are Grad(ient), Out(put). Noise Mechanisms include Dist(ributed) Lap(lace), Disc(rete) Gauss(ian), Poisson-Bin(omial). ✓/× denote existing/missing features, – gaps , and ⊛ potential enhancements. Privacy Unit is R(ecord) or U(ser) level. DP Analysis includes Mom(ents) Acc(ountant) and C(entral) L(imit) T(heorem) [60]. We indicate Malicious $\mathcal{C}/\mathcal{S}$, and Collusion thresholds.

Learning Paradigm	Type (Alg. 2)	Sampling (Fig. 3)	Noise Perturb (Alg. 1)	Mechanism (Tab. II)	Sampler	Gradient Secrecy $\mathcal{C}$	Gradient Secrecy $\mathcal{S}$	Model Secrecy $\mathcal{C}$	Model Secrecy $\mathcal{S}$	Oblivious Noise	Papers	Privacy Unit	DP Analysis	Cryptographic Techniques	Malicious	Collusion
FL	PNoise	Local	Grad.	Binomial	$\mathcal{C}$	×	⊛	×	⊛	⊛	[10]	U	(ϵ, δ)	Masking	×	×
				Disc. Gauss.	$\mathcal{C}$	×	⊛	×	⊛	⊛	[7]	U	RDP	Masking	×	×
					$\mathcal{C}$	×	⊛	×	⊛	⊛	[61]	R	RDP	Masking	×	×
					$\mathcal{C}$	×	⊛	×	⊛	⊛	[49]	R	RDP	LWE	$\mathcal{C}, \mathcal{S}$	$n/2$
				Skellam	$\mathcal{C}$	×	⊛	×	⊛	⊛	[62]	U	RDP	Masking	×	×
					$\mathcal{C}$	×	⊛	×	⊛	⊛	[63]	R	RDP	Masking	×	×
				Poisson-Bin.	$\mathcal{C}$	×	⊛	×	⊛	⊛	[64]	U	RDP	Masking	×	×
				Gaussian	$\mathcal{C}$	×	⊛	×	⊛	⊛	[65]	R	Mom. Acc.	(A)HE	×	$n-1$
					$\mathcal{C}$	×	✓	×	✓	⊛	[59]	U	Mom. Acc.	(F)HE	×	$n-1$
					$\mathcal{C}$	×	⊛	×	⊛	⊛	[66]	R	RDP	(A)HE	×	$n-1$
					$\mathcal{S}$	×	⊛	×	⊛	⊛	[67]	R	CLT	SS	$\mathcal{C}$	$n/2, m-1$
					$\mathcal{C}$	×	⊛	×	⊛	⊛	[68]	U	RDP	Masking	$\mathcal{C}, \mathcal{S}$	$n-1$
			Out.	Dist. Lap.	$\mathcal{C}$	✓	✓	×	⊛	✓	[69]	R	ϵ	Masking	×	$n-1$
					$\mathcal{C}$	✓	✓	×	⊛	✓	[11]	R	ϵ	A(HE)	×	$n-1$
					$\mathcal{C}$	✓	✓	×	⊛	✓	[70]	R	ϵ	A(HE)	$\mathcal{S}$	$n-1$
	CNoise	Distributed	Grad.	Gaussian	$\mathcal{S}$	×	⊛	×	⊛	✓	[71]	R	Mom. Acc.	SS + GC	×	×
					$\mathcal{S}$	×	⊛	×	⊛	✓	[8]	R	zCDP	SS + GC	$\mathcal{S}$	×
					$\mathcal{S}$	×	⊛	×	⊛	✓	[72]	R	(ϵ, δ)	SS + GC	×	×
			Out.	Laplace	$\mathcal{S}$	✓	✓	×	⊛	✓	[8]	R	ϵ	SS + GC	$\mathcal{S}$	×
		Centralized	Grad.	–	–	×	⊛	×	⊛	✓	–	–	–	–	–	–
			Out.	Laplace	$\mathcal{S}$	✓	✓	×	⊛	✓	[34]	R	ϵ	(A)HE	$\mathcal{C}, \mathcal{S}$	$m-1$
OL	PNoise	Local	Grad.	Binomial	$\mathcal{S}$	✓	✓	✓	✓	⊛	[9]	R	RDP	SS	×	$m/2$
				Disc. Gauss.	$\mathcal{S}$	✓	✓	✓	✓	⊛	[73]	R	Mom. Acc.	SS	×	$m-1$
			Out.	–	–	✓	✓	✓	✓	⊛	–	–	–	–	–	–
	CNoise	Distributed	Grad.	–	–	✓	✓	✓	✓	✓	–	–	–	–	–	–
			Out.	Dist. Lap.	$\mathcal{S}$	✓	✓	✓	✓	✓	[74]	R	ϵ	SS	$\mathcal{S}$	$m-1$
		Centralized	–	–	–	✓	✓	✓	✓	✓	–	–	–	–	–	–

detail noise obliviousness in ▶ Sec. V and its implications for CNoise in ▶ Sec. V-A and PNoise in ▶ Sec. V-B.

Privacy Unit. The granularity of DP's protection is typically given at user or record level. In CPCL the choice of privacy unit is crucial but often overlooked. Only 27% of systematized works provide user-level DP, i.e., where each user contributes up to z records. Notably, none of those works leverage OL. Protecting only individual records can harm users contributing multiple records. For example, in language modeling each user provides thousands of examples (i.e., words or phrases) [75]. We discuss how to achieve user-level DP in ▶ Sec. V.

DP Analysis. We specify the privacy accounting method to track the privacy budget across training iterations. Most of the systematized works leverage advanced accountants, e.g., moments accountant [37] or Rényi DP [36], for tighter privacy bounds. We provide definitions, conversion lemmas, and comparisons of composition bounds in ▶ App. C.

Cryptographic Techniques. HE and masking work with a single server but require key management. Specifically, HE requires key-pair generation via a trusted dealer or a distributed key generation protocol [65], while masking requires the exchange of pairwise masks (or seeds) [46]. In contrast, MPC requires multiple non-colluding servers but no keys, using only local randomness. The cryptographic techniques have different bottlenecks. MPC is communication-intensive, since servers exchange messages during computations. In comparison, HE is computation-heavy but communication-efficient as it requires only a single server. FL works employ all three techniques. However, masking is the most common likely since it is specific to FL. OL works rely mainly on MPC with optimized protocols. Notably, no OL work uses HE, even though it is beneficial in network-constrained settings, or hybrid HE-MPC approaches to balance computation and communication.

Threat Models. Most CPCL works assume *semi-honest security* with non-colluding parties for efficiency. Only few works consider *malicious security* against $\mathcal{S}$ and $\mathcal{C}$ during the whole training, i.e., noise sampling, gradient computation and aggregation [34], [49], or only during noise sampling [69] to defend against noise tampering. Notably, colluding parties can weaken privacy guarantees by removing their PNoise contributions from the aggregate. Mitigating this requires either increasing the noise variance (degrading accuracy) or cryptographic enhancements that introduce performance overhead. We analyze mitigations for collusion with PNoise in ▶ Sec. V-B, and evaluate the impact on accuracy in ▶ Tab. V, Sec. VI. Furthermore, we discuss privacy attacks and mitigations against malicious parties in ▶ App. P.

B. CPCL Framework

We propose a unified framework for CPCL by identifying seven common phases, detailed below for PerturbGradient.

Setup$(\epsilon, \delta, \Lambda, \mathcal{C}, \mathcal{S})$: $\mathcal{C}$ and $\mathcal{S}$ exchange cryptographic parameters Λ (e.g., keys, seeds) and DP parameters (ϵ, δ). Depending on the cryptographic technique, this may involve generating keys (HE), exchanging seeds (masking), or precomputing correlated randomness (MPC). Optionally, $\mathcal{C}$ may

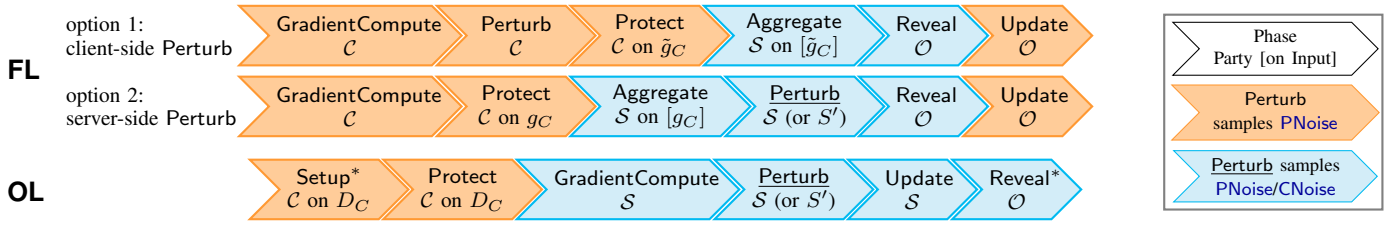


Fig. 1: Execution flows of FL and OL phases with PerturbGradient, party roles, inputs where relevant, and local or global computations. Setup is omitted, except OL-specific pre-processing. Optional phases have superscript $*$.

perform local data pre-processing (e.g., feature extraction) to improve training efficiency, which we discuss in ▶ Sec. IV-D.

GradientCompute($\theta^{(k)}$, D): $\mathcal{C}$ or $\mathcal{S}$ compute per-example gradients on data D at training step k. The per-example gradients $\mathbf{g}$ are clipped to bound sensitivity for DP noise. While trivial on cleartext data, clipping requires non-linear operations (e.g., inverse square root, comparisons) which are costly in cryptographic protocols. We discuss the cryptographic performance overhead of GradientCompute in ▶ Sec. IV-D and evaluate the performance impact in ▶ Tab. IV, Sec. VI.

Protect(x, $\mathcal{S}$): $\mathcal{C}$ encrypt data or gradients x to output encrypted $[x]_S$ to servers $S \in \mathcal{S}$. Specifically, in MPC clients secret share (Shr) data among m servers. Instead, in single-server settings, clients leverage HE or masking to encrypt (Enc) or mask the data creating a single ciphertext.

Perturb(τ, $\mathbf{g}$): $\mathcal{C}$ or $\mathcal{S}$ sample noise $\psi = \mathsf{Sample}(\tau)$ to output $\tilde{\mathbf{g}} = \mathbf{g} + \psi$. The tuple τ specifies the noise type (i.e., PNoise/CNoise), the sampling mechanism (Tab. II, Sec. V), and associated parameters (e.g., Gaussian PNoise with variance $\sigma^2 = 2\ln(1.25/\delta)\Delta_2^2/\epsilon^2$ to satisfy (ϵ, δ)-DP). Notably, we stylize sampling options as: Perturb for local PNoise sampling, Perturb for CNoise via MPC, and Perturb for CNoise via a semi-trusted server. Perturb indicates any of these three may be used. We identify Perturb as the foundational phase for integrating DP in cryptographic CL, and analyze all its components in depth in ▶ Sec. V. We formalize Sample algorithms in ▶ Alg. 3 (App. J-A).

Aggregate($\mathbf{G}$): $\mathcal{S}$ aggregates the set $\mathbf{G}$ of locally computed gradients, typically via averaging. Specific to FL, this phase relies on lightweight operations on encrypted data (e.g., additions) as further discussed in ▶ Sec. IV-C.

Update($\theta^{(k)}$, $\tilde{\mathbf{g}}$, η): $\mathcal{C}$ or $\mathcal{S}$ update model parameters $\theta^{(k)}$ at step k using noisy gradients $\tilde{\mathbf{g}}$ and learning rate η.

Reveal($\{[x]_S\}_{\mathcal{S}}$): $\mathcal{C}$ or $\mathcal{S}$ decrypt cleartext x from encrypted $[x]_S$ received from $S \in \mathcal{S}$ (i.e., via Rec or Dec). We distinguish two types of reveal: *implicit* which reveals results to $\mathcal{S}$ immediately upon aggregation, offering no gradient secrecy against $\mathcal{S}$ (e.g., pair-wise masking). *Explicit reveal* requires a distinct decryption protocol and if performed by $\mathcal{C}$ guarantees gradient and model secrecy against $\mathcal{S}$ (e.g., MPC). We discuss how reveal affects security of FL in ▶ Sec. IV-C.

We provide cryptographic computation and communication complexities across phases in ▶ Tab. VIII, App. E-D.

Framework Instantiations. Our framework is general and covers different designs. This is achieved by adjusting the order of phases, the set of parties executing them (i.e., clients $\mathcal{C}$, servers $\mathcal{S}$, and output parties $\mathcal{O} \subseteq \mathcal{C} \cup \mathcal{S}$), and the type of computations, i.e., local on cleartext data or global on encrypted data. Fig. 1 provides a high-level overview of how to instantiate FL and OL with our framework, which we detail in ▶ Sec. IV-C, IV-D, respectively. Although we mainly focus on FL and OL, our framework can easily express other learning paradigms, e.g., split learning [24], and vertical FL [22], as detailed in ▶ App. F (see Fig. 2). Furthermore, we formalize the phases as pseudocode in Alg. 2. Specifically, phases are at the top and execution flow of FL and OL are given and compared at the bottom. Overall, we can model single-server FL (e.g., via HE) by setting $|\mathcal{S}| = 1$, where each $C \in \mathcal{C}$ sends one ciphertext to a server S, i.e., $[\tilde{\mathbf{g}}_C]_S$ for PNoise and $[\mathbf{g}_C]_S$ for CNoise. By setting $|\mathcal{S}| > 1$, we model multi-server FL (via MPC) where each $C \in \mathcal{C}$ sends shares, e.g., $[\tilde{\mathbf{g}}_C]_S$, to each $S \in \mathcal{S}$. Additionally, we can also model privacy amplification by *subsampling* [76] (App. D): in FL a random subset of clients are sampled for each iteration, whereas in OL, subsampling on D can be implemented within GradientCompute. Also, to model PerturbOutput, Perturb is applied on model parameters *after* training, without clipping.

C. Federated Learning

Fig. 1 identifies two design options for FL: client-side and server-side Perturb. Independently of the design choice, clients $\mathcal{C}$ iteratively compute DP-SGD's per-example clipped gradients $\mathbf{g}_C$ via GradientCompute locally. In client-side Perturb, clients $\mathcal{C}$ sample local PNoise via Perturb before Protect. After receiving the encrypted noisy local updates, servers $\mathcal{S}$ merge them in Aggregate. Instead, in server-side Perturb, $\mathcal{C}$ only execute Protect, while $\mathcal{S}$ perform Aggregate and Perturb, sampling PNoise locally or CNoise globally. In both option, output parties $\mathcal{O}$ receive noisy aggregated gradients $\tilde{\mathbf{g}}$ and perform Update locally. Before Protect, $\mathcal{C}$ quantize updates to representations suitable for encrypted computations, e.g., fixed-point. Reveal decrypts (and dequantizes) the noisy aggregated $[\tilde{\mathbf{g}}]_s$. We detail quantization approaches in App. H.

Privacy Guarantees. FL does not inherently guarantee gradient and model secrecy against $\mathcal{C}$, since $\mathcal{C}$ locally compute gradients on cleartext data. However, gradient and model secrecy can be guaranteed against $\mathcal{S}$ by leveraging cryptographic techniques. Specifically, using explicit Reveal performed by $\mathcal{C}$ (i.e., $\mathcal{O} = \mathcal{C}$), e.g., via HE, MPC, or LWE-based masking with distributed Dec/Rec. Additionally, FL has to account for dropouts, i.e., clients leaving during training, which can disrupt

Input: Sets of clients $\mathcal{C}$, servers $\mathcal{S}$, and output parties $\mathcal{O}$ where $\mathcal{C} \subseteq \mathcal{O} \subseteq \mathcal{C} \cup \mathcal{S}$ (i.e., we assume all clients are output parties, servers optionally); clipping parameter K, per client training dataset $D_C = \{x_i, y_i\}_{i=1}^{N_C}$ with size N_C, model parameters $\theta^{(k)}$ at iteration k.
Notation: Cleartext phases apply to encrypted data when executed globally. τ in Perturb defines noise type (PNoise/CNoise), mechanism (Tab. II) and distribution parameters (e.g., σ^2 for (ϵ, δ)-DP). PNoise is sampled locally via Perturb, while CNoise is sampled via MPC Perturb or by semi-trusted S' (Perturb). $\{\}_S$ denotes a set for each $S \in \mathcal{S}$.
Output: Updated model parameters $\theta^{(k+1)}$.

Function Clip $(\mathbf{G}, K)$:
  return $\{\mathbf{g} \cdot \min(1, K/||\mathbf{g}||_2)\}_{g \in \mathbf{G}}$

Phase Perturb $(\tau, \mathbf{g})$:
  $\psi \leftarrow$ Sample(τ) `// See Alg. 3`
  return $\mathbf{g} + \psi$

Phase GradientCompute $(\theta^{(k)}, D, K)$:
  $\mathbf{L}(\theta^{(k)}, D) \leftarrow \{\ell(\theta^{(k)}, x_i, y_i)\}_{i \in [1,N]}$
  $\mathbf{G} \leftarrow \nabla_\theta \mathbf{L}(\theta^{(k)}, D)$
  $\mathbf{G}' \leftarrow$ Clip$(\mathbf{G}, K)$
  return $\frac{1}{|\mathbf{G}'|} \sum_{\mathbf{g}' \in \mathbf{G}'} \mathbf{g}'$

Phase Protect $(\mathbf{x}, \mathcal{S})$:
  return $\{[\mathbf{x}]_S\}_S$

Phase Reveal $(\{[x]_S\}_S)$:
  return $\mathbf{x}$

Phase Aggregate $(\mathbf{G})$:
  return $\frac{1}{|\mathbf{G}|} \sum_{\mathbf{g} \in \mathbf{G}} \mathbf{g}$

Phase Update $(\theta^{(k)}, \tilde{\mathbf{g}}, \eta)$:
  return $\theta^{(k)} - \eta \cdot \tilde{\mathbf{g}}$

FEDERATED LEARNING:
foreach $C \in \mathcal{C}$ **do** `// For each training iteration` k
  $\mathbf{g_c} \leftarrow$ GradientCompute $(\theta_c^{(k)}, D_c, K)$
  `// Client-side noise sampling`
  $\tilde{\mathbf{g}}_\mathbf{c} \leftarrow$ Perturb $(\tau, \mathbf{g_C})$
  $\{[\tilde{\mathbf{g}}_\mathbf{C}]_S\}_S \leftarrow$ Protect $(\tilde{\mathbf{g}}_\mathbf{C}, \mathcal{S})$
  for $S \in \mathcal{S}$ **do** send $[\tilde{\mathbf{g}}_\mathbf{C}]_S$ to S
foreach $S \in \mathcal{S}$ **do**
  `// Server-side noise sampling`
  $\{[\mathbf{g_C}]_S\}_S \leftarrow$ Protect $(\mathbf{g_C}, \mathcal{S})$
  for $s \in \mathcal{S}$ **do** send $[\mathbf{g_C}]_s$ to S
foreach $S \in \mathcal{S}$ **do**
  $[\mathbf{g}]_S \leftarrow$ Aggregate $(\{[\mathbf{g_C}]_S\}_\mathcal{C})$
  $[\tilde{\mathbf{g}}]_S \leftarrow$ Perturb$(\tau, [\mathbf{g}]_S)$
  for $O \in \mathcal{O}$ **do** send $[\tilde{\mathbf{g}}]_S$ to O
foreach $O \in \mathcal{O}$ **do**
  $\tilde{\mathbf{g}} \leftarrow$ Reveal $(\{[\tilde{\mathbf{g}}]_S\}_S)$
  $\theta_O^{(k+1)} \leftarrow$ Update $(\theta_O^{(k)}, \tilde{\mathbf{g}}, \eta)$

OUTSOURCED LEARNING:
foreach $C \in \mathcal{C}$ **do** `// Only first iteration`
  $\{[D_C]_S\}_S \leftarrow$ Protect$(D_C, \mathcal{S})$
  for $S \in \mathcal{S}$ **do** send $[D_C]_S$ to S
foreach $S \in \mathcal{S}$ **do** `// For each training iteration` k
  $[\mathbf{g}]_S \leftarrow$ GradientCompute $([\theta^{(k)}]_S, \{[D_C]_S\}_\mathcal{C}, K)$
  `// Server-side noise sampling`
  $[\tilde{\mathbf{g}}]_S \leftarrow$ Perturb $(\tau, [\mathbf{g}]_S)$
  $[\theta^{(k+1)}]_S \leftarrow$ Update $([\theta^{(k)}]_S, [\tilde{\mathbf{g}}]_S, \eta)$
foreach $S \in \mathcal{S}$ **do** `// Only last iteration` E `(Optional)`
  for $O \in \mathcal{O}$ **do** send $[\theta^{(E)}]_S$ to O
foreach $O \in \mathcal{O}$ **do**
  $\theta_O^{(E)} \leftarrow$ Reveal $(\{[\theta^{(E)}]_S\}_S)$

Algorithm 2: Comparison of FL and OL phases per training iteration using PerturbGradient with PNoise or CNoise.

Reveal if all clients are needed, and can introduce privacy risks, e.g., leaking local updates as discussed in App. I.

Performance. In FL, $\mathcal{C}$ are actively involved executing GradientCompute and Protect locally. While naive per-example clipping can increase GradientCompute runtime by up to $10\times$, cleartext optimizations, e.g., ghost clipping [77], effectively reduce overhead. The main bottleneck in FL is Aggregate, which combines local updates from $\mathcal{C}$ introducing a communication bottleneck, i.e., up to 95% of total runtime in our evaluation (Tab. IV, Sec. VI). Here, $\mathcal{C}$ iteratively send encrypted messages of size $O(|\theta|)$ to $\mathcal{S}$. For HE and masking each client sends $O(1)$ messages; with MPC each client sends $O(m)$ messages. Instead, $\mathcal{S}$ perform $O(n)$ local, i.e., interaction free, additions for all techniques, and send back the aggregated result, optionally performing Reveal. For details on cryptographic techniques for SecAgg, we refer to SoK [17].

D. Outsourced Learning

Fig. 1 show that clients $\mathcal{C}$ apply Protect on their local data, while servers $\mathcal{S}$ run GradientCompute globally. OL supports only server-side Perturb on encrypted gradients. Here, $\mathcal{S}$ execute Perturb, which can locally sample PNoise; or sample CNoise via cryptographic protocols or via a semi-trusted S'. Finally, $\mathcal{S}$ run Update, aggregating PNoise. Optionally, clients can pre-process local data in Setup, and encrypt the result.

Privacy Guarantees. By design OL does not require Aggregate and Reveal, unlike FL. Thus, **(O1) OL inherently guarantees gradient and model secrecy against both clients and servers,** preventing GIA. Clients can opt to reveal the final DP model to enable local inference without cryptographic overhead. Additionally, for strong convex losses, hiding intermediate updates improves DP composition bounds [78], potentially offering tighter guarantees than FL.

Performance. In OL, clients are not actively involved during the whole learning process. They execute Protect only once and can go offline, unlike FL. Here, servers execute GradientCompute over encrypted data $[D]_S$ and model parameters $[\theta]_S$ via cryptographic protocols. This requires costly approximations for non-linear functions, e.g., activations and comparisons, which introduce an accuracy-efficiency trade-off and can make OL up to $30\times$ slower than FL in non-DP settings (Tab. IV, Sec. VI). For a comprehensive overview of cryptographic OL solutions, we refer to SoK [18]. Additionally, DP-SGD requires per-example gradient clipping. While FL clients perform Clip locally and can leverage cleartext optimizations (e.g., ghost clipping [77]), in OL cryptographic Clip presents a dual challenge. For OL, **(O2) there are no secure per-example clipping optimizations via cryptographic protocols.** Also, Clip requires computation of the inverse of the square root, multiplications and comparison, incurring significant overhead, i.e., up to 50% of total runtime (Tab. IV, Sec. VI). While recent advancements exist [79], [80], only PEA [9] proposes an optimized protocol for the inverse of square root based on polynomial approximations in CPCL. Notably, GradientCompute performance can be improved indirectly, as **(O3) client local pre-processing in OL reduces cryptographic overhead while maintaining high accuracy.** Here, clients can perform, e.g., dimensionality reduction, feature extraction, or train local models to serve as an initial global model [9]. Thus, servers can train simpler models, like logistic regression, on extracted features instead of complex models, such as neural networks (NN), on raw data. For example, PEA [9] effectively combines these techniques, achieving a $100\times$ speedup by training a logistic regression on CIFAR-10 with the accuracy of a CNN [81].

V. NOISE SAMPLING TECHNIQUES

This section provides a comprehensive analysis of Perturb, the foundational phase of CPCL. We first overview common design choices, then detail the sampling techniques for PNoise and CNoise (Sec. IV-A) by increasing cryptographic overhead.

TABLE II: DP noise distributions. Random variables are denoted like PDFs without dependence on x.

Distribution	Probability Density Function (PDF)
Laplace	$\mathsf{Lap}(x;\lambda) = \frac{1}{2\lambda}\exp(\lvert x\rvert/\lambda)$
Dist. Laplace	$\mathsf{Lap}(\lambda) = \sum_{k=1}^{N} g_k^1 - g_k^2;\ g_k^1, g_k^2 \sim \mathsf{Gamma}(x;\frac{1}{N},\lambda)$
Gaussian	$\mathcal{N}(x;\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp(-x^2/2\sigma^2)$
Binomial	$\mathsf{Bin}(x;m,p) = \binom{m}{x}p^x(1-p)^{m-x}$
Disc. Gaussian	$\mathcal{N}_{\mathbb{Z}}(x;\sigma^2) = e^{-x^2/2\sigma^2} / \sum_{y\in\mathbb{Z}} e^{-y^2/2\sigma^2}$
Skellam	$\mathsf{Sk}(\mu) = p_1 - p_2;\ p_1, p_2 \sim \mathsf{Poisson}(x;\mu/2)$
Poisson-Binomial	$\mathsf{PoiBin}(x;\mathbf{g},b,p) = \mathsf{Bin}(x;b,\mathbf{g}\theta/K + 1/2)$

Noise Mechanisms. We discuss the suitability of noise distributions in Tab. II for noise types, perturbation options and DP guarantees. We formalize sampling algorithms (Sample) in Alg. 3 (App. J-A) and provide conversion between DP and distribution parameters in App. J. Note that all noise mechanisms are model agnostic and the choice between different mechanisms is based on analytical properties of the distributions rather than model architecture. Tab. I shows that most works use Gaussian-like mechanisms for PerturbGradient, and Laplace for PerturbOutput. Laplace is applied once in PerturbOutput providing pure DP ($\delta = 0$). In contrast, PerturbGradient applies noise iteratively, where Gaussian is preferred as it is closed under summation, i.e., $\sum_{i=1}^{N}\mathcal{N}_i(\mu_i,\sigma_i^2) = \mathcal{N}(\sum_{i=1}^{N}\mu_i, \sum_{i=1}^{N}\sigma_i^2)$. Moreover, Gaussian has faster tail decay than Laplace, and the noise scales with the l_2-sensitivity, often smaller than the l_1 for Laplace [36]. Both noise mechanisms can be adapted to be aggregated from PNoise (Sec. V-B): Laplace via the difference of N i.i.d. Gamma random variables [11], [69], and Gaussian leveraging its closure under summation. Notably, secure sampling of continuous noise typically requires quantization as cryptography uses fixed-point representations [10]. Alternatively, discrete distributions enable sampling from discrete domains. Next, we review discrete distributions that mimic Gaussian guarantees for PerturbGradient, which systematized works apply for PNoise [10], [62], [82]. The *binomial* mechanism [9], [10] satisfies (ϵ,δ)-DP, and it is closed under summation, i.e., $\mathsf{Bin}(m,p) = \sum_i^n \mathsf{Bin}(m_i,p)$. The sum of *Discrete Gaussian* [7] satisfies (α,ϵ)-RDP, despite not being closed under summation. The *Skellam* mechanism [62], [63], realized as the difference of two independent Poisson random variables, satisfies (α,ϵ)-RDP and is closed under summation. Note that all mechanisms in Tab. II add noise to gradients except for Poisson-Binomial [64] which encodes local gradients into a parameter of the binomial and outputs a discrete noisy value.

Obliviousness. We distinguish noise obliviousness at selection and sampling level. *Oblivious selection* [34] can be realized by a client or server locally sampling a list of candidate noises, encrypting and shuffling them; then another party selects a value from the encrypted list. *Oblivious sampling* [8] requires the noise to be sampled via cryptographic protocols, ensuring it remains encrypted and hidden from all parties.

Privacy Unit. While for record-level DP, $\mathcal{C}$ or $\mathcal{S}$ clip per-record gradients, user-level DP requires bounding the contribution of all records from a user. In FL, each user holds only its own data and the record-level guarantee can be extended to user level by clipping local model updates as in FedAvg [75]. Instead, in OL, local datasets are pooled in a global dataset. Here, $\mathcal{S}$ need to securely identify user records and bound per-user contribution via cryptographic protocols. Furthermore, **(O4) no established best practice exists for user-level DP in multi-user datasets.** A straightforward way is to emulate FedAvg in OL: servers sample records per user from the global dataset, and then compute and clip per-user gradients [83]. Alternatively, DP satisfies *group privacy* enabling the translation from record- to user-level DP: data containing at most z records per user satisfies user-level $(z\epsilon, z\exp(z\epsilon)\delta)$-DP [31]. However, this requires strict contribution bounding and rapidly weakens privacy guarantees for large z.

A. CNoise: *Centralized Sampling*

In centralized sampling, a semi-trusted server S' samples CNoise ψ in clear. This noise is then added to the encrypted aggregated gradient, either by sending encrypted noise to the computing servers (Perturb) or by adding it directly if S' is oblivious to the gradient values.

Rationale & Trade-offs. This technique offers the optimal privacy-accuracy trade-off, as in CDP, since noise is added once to global data, e.g., aggregated gradients in FL. Computationally, it is highly efficient because sampling occurs in clear. In our evaluation (Sec. VI), we estimate this adds negligible runtime overhead to OL and small latency to FL ($\approx 10\%$ runtime increase due to communication). However, it introduces a strong trust assumption: S' can learn either the noisy output $\tilde{\mathbf{g}}$ or the noise ψ, *never* both.

Obliviousness Considerations. Obliviousness to the noisy output $\tilde{\mathbf{g}}$ is enforced with different techniques, depending on the learning paradigm and the set of output parties $\mathcal{O}$. In FL, if $\mathcal{S} \subset \mathcal{O}$, separation of duties can be applied: one server $S' \notin \mathcal{S}$ can sample and encrypt CNoise ψ, then distribute the encrypted noise term $[\psi]_s$ to each $S \in \mathcal{S}$ for decentralized noise addition. Alternatively, one server $S' \notin \mathcal{S}$ samples and adds CNoise ψ to the encrypted gradients $[\mathbf{g}]_s$, while $\mathcal{S}$ decrypt the noisy aggregate $\tilde{\mathbf{g}}$. The latter solution suits, e.g., HE-based FL, where a decryption key is required. Instead, MPC-based solutions require additional masking or re-sharing to prevent S' from learning the gradients [84]. In OL, servers only see random-looking, encrypted data, i.e., $\mathcal{S} \not\subseteq \mathcal{O}$, inherently guaranteeing obliviousness to $\tilde{\mathbf{g}}$. Thus, a server $S' \in \mathcal{S}$ can sample and add CNoise ψ to its local encrypted gradients $[\mathbf{g}]_{s'}$.

Noise obliviousness can be achieved via oblivious noise selection, regardless of the learning paradigm. Here, S' samples and encrypts a list of l CNoise terms. The list is then shuffled, one term is obliviously selected and added to the encrypted gradients. For example, Drynx's [34] servers run a verifiable shuffle protocol [85], and select the first encrypted CNoise in the shuffled list. The probability of guessing the chosen noise decreases linearly with l, i.e., $\delta \propto 1/l$.

Obliviousness Overhead. With output obliviousness ($S' \notin \mathcal{O}$), S' either locally adds noise over encrypted data, or distributes encrypted noise $[\psi]_s$ in $O(m)$ messages. Noise

obliviousness via oblivious selection introduces a privacy-performance trade-off: increasing the number l of noise terms enhances privacy, but also increases cryptographic overhead as computation and communication scale linearly with l [34].

B. PNoise: Local Sampling And Aggregation

In partial noise aggregation, each client $C \in \mathcal{C}$ locally samples non-DP PNoise ψ_C. Notably, while ψ_C alone does not satisfy DP, their sum $\sum_{C \in \mathcal{C}} \psi_C$ satisfies CDP. As Alg. 2 shows, in FL, clients add ψ_C on local gradients $\mathbf{g_C}$ and then Protect ensures local gradient remain hidden from servers. Alternatively, each server $S \in \mathcal{S}$ adds PNoise ψ_S on local encrypted gradients $[\mathbf{g}]_S$ in both FL and OL.

Rationale & Trade-offs. PNoise is the dominant technique in FL as it allows local sampling in cleartext, avoiding cryptographic protocols. In our evaluation, the runtime overhead is only around 2% for FL and negligible in OL (Tab. IV, Sec. VI). To satisfy (ϵ, δ)-DP, a specific variance σ^2_{DP} is required (App. J). For example, each $C \in \mathcal{C}$ (or $S \in \mathcal{S}$), samples Gaussian noise with variance $\sigma^2_C \geq \sigma^2_{\text{DP}}/n$. Importantly, in FL, noise-sampling clients learn aggregated noisy updates $\tilde{\mathbf{g}} = 1/n \sum_{C \in \mathcal{C}} (\mathbf{g}_C + \psi_C)$. A passive adversary corrupting a client, or up to t colluding clients, can remove their noise ψ_C from the noisy aggregated $\tilde{\mathbf{g}}$, thereby weakening the targeted privacy guarantee and potentially launching a GIA (App. P). Thus, each client must account for potential collusion by choosing a higher minimum variance σ^2_C adjusted for t:

$$\sigma^2_C \geq \frac{1}{n-t} \sigma^2_{\text{DP}}. \tag{3}$$

The adjusted variance ensures $\tilde{\mathbf{g}}$ satisfies CDP even if t colluding clients remove their noise. With large n (e.g., 100) and high t (e.g., $n-1$), total noise increases beyond LDP variance, degrading accuracy to near random guessing (Tab. V, Sec. VI). To also tolerate up to s dropouts, the denominator of Eq. (3) can be set to $n-(t+s)$. Most PerturbGradient-based FL works do not account for collusion (Tab. I), since it either increases the total noise variance (degrading accuracy) or introduces cryptographic overhead, as discussed below.

Oblivious Selection. To reduce the noise variance in Eq. (3), **(O5) local PNoise sampling can be enhanced with noise obliviousness.** We distinguish *client-side* and *server-aided* protocols. In client-side protocols, each client receives l encrypted PNoise terms and obliviously selects one. The clients can obtain noise terms from topological neighbors [69], or a server can shuffle and forward l encrypted noises per client [11]. For the latter, t colluding clients do not know who generated their noise but may still average their noise contributions to weaken the DP guarantee. However, even $l = 2$ and $t = n-1$ is not enough to infer honest clients' private information [11]. Alternatively, in server-aided protocols, clients and servers jointly select noise terms. Specifically, each client locally samples and encrypts l PNoise terms. The server generates and encrypts an l-bit selection vector to select one PNoise per client (i.e., with 1 for selection, 0 for omission). The encrypted selection vector and noise terms are multiplied element-wise to produce oblivious PNoise samples. For example, the server sends HE-encrypted selection vectors to clients, which are multiplied with l encrypted PNoise [70].

Obliviousness Selection Overhead. Unlike local PNoise sampling, oblivious noise selection incurs cryptographic overhead. In client-side protocols [11], [69], each client samples l noise terms for any other client or a subset of neighboring clients, resulting in $O(nl)$ local sampling and encryptions. The encrypted noise can be sent to a central server acting as a relay which can batch the encrypted lists in $O(n)$ messages [11], or exchanged between clients in $O(n^2)$ messages [69]. Computation and communication costs scale linearly in n and l. Runtime is mainly dominated by network latency, though its impact diminishes as n increases [11]. Server-assisted protocols [70] reduce communication cost to $O(n)$ messages containing l selection bits. Due to the cryptographic overhead, only PerturbOutput works use oblivious selection [11], [69], [70], as the oblivious protocol is run once.

Noise Mechanisms Considerations. (O6) The choice of noise distribution affects utility and performance. Distributions not closed under summation can diverge significantly at small noise levels, i.e., large number of clients [62]. For discrete Gaussian with $n = 10^4$ and $\sigma_i = 0.5$, the RDP bound is $10^6\times$ larger than for Skellam [62]. For the binomial [10], we are not aware of advanced accounting methods (e.g., moments accountant), which may result in injecting suboptimal noise. Currently, ML libraries [86], [87] support efficient sampling for certain distributions, e.g., Gaussian and Skellam, but not for the discrete Gaussian [7] which can be up to 40% slower than Skellam [62]. Among discrete distributions, Skellam is the most efficient and accurate choice for PNoise reaching the accuracy of continuous Gaussian using half-precision (16 bits) [62]. Notably, inherent errors from cryptographic approximations and quantization can serve as DP noise, suggesting that **(O7) DP can be embedded in phases besides Perturb,** i.e., in Protect or during quantization. Non-additive mechanisms, like Poisson-Binomial [64], directly map continuous gradients to discrete noisy values. Also, non-additive discrete mechanisms have bounded support and their communication costs decrease with ϵ [64]. Remarkably, noise-based cryptographic techniques can integrate DP without sampling additional noise. For example, in LWE-based masking [49] a noise vector e breaks the linearity of a set of equations. The vector e itself can guarantee DP, e.g., by being sampled from a discrete Gaussian distribution [49]. Thus, the LWE decryption does not remove all noise, as typically desired, but DP-suitable noise remains.

C. CNoise: Distributed Sampling

In distributed sampling, servers $\mathcal{S}$ collaboratively run Perturb to sample CNoise $[\psi]$ via cryptographic protocols, e.g., MPC. As Alg. 2 shows, each $S \in \mathcal{S}$ adds a noise share $[\psi]_S$ to the local encrypted $[\mathbf{g}]_S$.

Rationale & Trade-offs. This approach provides the best accuracy and security guarantees. It neither requires a semi-trusted S' as in centralized CNoise, nor increases noise variance to handle collusion as in PNoise. However, sampling via

cryptographic protocols incurs significant performance overhead. Our evaluation shows that generating a single Gaussian sample via MPC is about $10^3\times$ slower than local sampling (Tab. IV, Sec. VI). While this overhead is small for OL it increases FL runtime by nearly $10\times$, making it impractical.

MPC Sampling Challenges. Basic continuous sampling algorithms like Inverse Transform Sampling [88] and Box-Muller [89] enable Laplace and Gaussian mechanisms in MPC by transforming uniform samples $u \sim U(0,1)$ via fixed arithmetic operations (e.g., $\log$, $\sin$) [8], [71], [74] as detailed in Alg. 3, App. J-A. To generate a truly random u, servers can sample and combine local values, e.g., via *XOR* [8]. Recent works [55], [90]–[92] explore MPC sampling of discrete Laplace and Gaussian distributions, but not in CPCL (gaps – in Tab. I). Typically, CPCL uses quantized representations with fixed bit-width, e.g., fixed point, which require scaling the noise variance according to the quantization scale s, to avoid output only multiple of s [62]. This leads to unexplored high variance affecting performance and accuracy. Note that while continuous Gaussian scales with σ^2 and μ, i.e., $\sigma \cdot \mathcal{N}(0,1) + \mu = \mathcal{N}(\mu, \sigma^2)$, this does not hold for discrete Gaussian [93]. Additionally, no existing work samples Skellam via MPC [62]. A key challenge is that often **(O8) sampling algorithms for discrete distributions have non-constant runtimes** due to rejection sampling, which samples iteratively until a specific condition is met. This introduces data-dependent loops, e.g., **while**, whose runtime leaks information on the noise values, e.g., the loop count in Poisson reveals the sampled noise p (Alg. 3). To prevent leaks, loops with a fixed iteration count, e.g., **for**, must replace data-dependent ones. The iteration count is set to ensure, except with negligible failure probability, that at least one correct sample is returned. Additionally, fixed bit-width can lead to underflow or overflow, e.g., $\exp(-1/\mu)$ in Poisson can cause underflow for large μ.

MPC Sampling Overhead. The runtime of iterative algorithms for discrete noise sampling in MPC depends mainly on the fixed iteration count set to ensure constant-time execution, which is affected by DP parameters, bit-width, and failure probability [55], [91]. To evaluate the overhead, we implement MPC versions of DiscGauss [7] and Skellam [94] (Alg. 3). We scale the variance of DiscGauss ($s^2\sigma^2 = 29698$) and Skellam ($s^2\mu = 30840$) for $\epsilon = 1$, $\delta = 10^{-6}$ and $K = 0.1$, according to [7], [62]. We empirically set the iteration count to satisfy a failure probability $< 10^{-6}$ (App. N-A). We evaluate in WAN (Tab. III) and LAN (App. N-B) using the MPC framework MP-SPDZ [95], with setup details in App. L. We select three SS schemes: 3-party semi-honest and malicious Shamir with honest-majority ($t < m/2$), and 2-party malicious Mascot [96] with dishonest-majority ($t > m/2$). Tab. III reports average runtime over 10 runs with 95% confidence intervals. To address the underflow in Skellam for $\exp(-1/\mu)$ with large μ, we leverage Skellam's closure under summation, by sampling and summing 1542 Skellam samples with small $\mu = 20$. Skellam's runtime depends on μ and requires $2(\mu+1)$ iterations on average. From our results, semi-honest and malicious Shamir have comparable times, whereas Mascot is at least $2\times$ slower. Notably, discrete sampling algorithms are at least $10\times$ slower than continuous ones due to their iterative nature, potentially explaining the gap in Tab. I on distributed discrete CNoise sampling. We omit Skellam in Tab. III, as its runtime is disproportionate, i.e., about 20 minutes for all schemes in a LAN. LAN runtimes are $10\times$ faster than WAN, with online/offline phase and communication detailed in App. N. Overall, Laplace is the fastest but mainly used for PerturbOutput (Sec. V). Gaussian via BoxMuller is the most efficient for PerturbGradient, generating two samples per run.

TABLE III: Distributed CNoise sampling runtime for WAN.

WAN, seconds	Laplace	Gaussian	DiscGauss
Shamir	6.21 ± 0.03	18.57 ± 0.13	232.59 ± 0.36
Malicious Shamir	7.36 ± 0.03	18.76 ± 0.09	261.64 ± 0.44
Mascot	34.91 ± 0.54	107.2 ± 2.11	1638.12 ± 8.80

VI. Empirical Evaluation & Design Trade-Offs

Next, we evaluate key trade-offs and considerations to enhance cryptographic CL with DP. Specifically, we analyze DP's impact on runtime (Tab. IV) and accuracy (Tab. V) across paradigms and noise techniques. Using standard benchmarks (MNIST, Fashion MNIST) and setups from prior work [37], [73], we provide representative empirical evidence on the practical implications of design choices in CPCL.

Evaluation Setup. We train a 3-layer NN ($\approx$ 90K parameters and ReLU activations) on MNIST and Fashion MNIST datasets, as in [37], [73]. Training runtimes (Tab. IV) are averaged over 10 runs for batch size B, split into *mini-batches* of size b for efficiency with gradient accumulation. We evaluate the impact of naive Clip with $b = 1$ and Perturb compared to non-DP training with $b = 1$ and $b > 1$ to estimate the speed-up of batching. For accuracy (Tab. V), we vary $\epsilon \in \{1, 3, 8\}$, $\delta = 10^{-5}$, use Gaussian noise $\mathcal{N}(0, (\sigma K)^2)$, where $\sigma \in \{1.18, 0.73, 0.54\}$, and evaluate collusion thresholds $t \in \{n/2, n-1\}$ for PNoise. For single-server FL, we select $n = 100$ clients per round from 1000, and hyperparameters according to our search (App. O-A). For OL, we use hyperparameters from [37]. We implement FL with PFL [97]. For OL, we extend CrypTen [98] with missing DP-SGD components. App. L details the full evaluation setup.

Clipping Impact. Naive per-example clipping ($b = 1$) significantly increases runtime and memory usage by computing (and clipping) B gradients instead of one per batch (Tab. IV). In FL, $b = 1$ increases non-DP GradientCompute runtime by $8\times$, and Clip makes DP GradientCompute $2\times$ slower. However, Aggregate, which accounts for 75% of DP FL runtime due to communication and encrypted aggregation overhead, minimizes the impact of non-batched computation, keeping FL runtimes in the tens-of-seconds range. Instead, MPC-based inverse square root and comparisons in Clip make DP OL $2\times$ slower than non-DP OL ($b = 1$). Here, non-batched computations increase non-DP runtime by $100\times$. Notably, accuracy depends on K: a large K increases noise variance, obscuring the gradients, while a small K hinders convergence.

TABLE IV: Runtime of FL/OL over a batch in LAN, with ⌊ denoting the subprotocols of the above runtime.

Runtime (seconds)	FL ($B = 60, n = 100$)	OL ($B = 500, m = 2$)
Total non-DP ($b_{\text{FL}} = 10, b_{\text{OT}} = 128$)	0.19 ± 0.01	6.49 ± 0.05
⌊ GradientCompute non-DP	0.0047 ± 0.0008	6.45 ± 0.04
⌊ Aggregate	0.18 ± 0.011	–
Total non-DP ($b = 1$)	0.22 ± 0.0149	627.73 ± 3.38
⌊ GradientCompute non-DP	0.036 ± 0.004	627.72 ± 3.38
Total DP ($b = 1$) with PNoise	0.25 ± 0.024	1236.81 ± 55.12
⌊ GradientCompute DP	0.060 ± 0.0130	1236.79 ± 55.12
⌊ Clip ($K_{\text{FL}} = 0.3, K_{\text{OL}} = 4.0$)	0.023 ± 0.0043	718.93 ± 32.55
⌊ Perturb (PNoise, Gaussian)	0.0051 ± 0.0007	
Total DP ($b = 1$) with CNoise	3.11 ± 0.125	1238.87 ± 55.23
⌊ Perturb (CNoise, Gaussian)	2.87 ± 0.10	

For example, in OL, $K = 3$ yields 82% accuracy on EMNIST, $K = 1$ drops accuracy by 3 percentage points (pp), while $K = 10$ results in near-random guessing. (App. O-B). In PNoise, K needs to account for increased noise variance from collusion, e.g., a sub-optimal $K = 0.5$ leads to a 17 pp accuracy drop on MNIST in FL, vs. $K = 0.3$ ($\epsilon = 3, t = n - 1$).

Noise Techniques. Our evaluations (Tab. IV, Tab. V) highlight the performance-accuracy trade-offs of different noise sampling techniques. PNoise sampling is the most efficient secure sampling technique, as it relies on local sampling, with a minimal impact on FL runtime (2%) and negligible on OL ($\ll$ 1%). However, PNoise's utility is compromised under high collusion ($t = n - 1$), reducing accuracy by up to 70 pp (Tab. V). Mitigating this requires cryptographic enhancements: oblivious protocols for PNoise, or sampling CNoise, which introduce performance overhead. While centralized CNoise sampling achieves optimal utility (as in CDP), it requires a semi-trusted S' and obliviousness (Sec. V-A). If $S' \notin \mathcal{O}$, communication latency to send encrypted noises is the main overhead, which is negligible for OL, but more significant for FL (10% of PNoise runtime). Distributed CNoise sampling removes the need for S' and maintains optimal utility. However, it incurs large MPC sampling overhead, e.g., MPC sampling of Gaussian noise is almost $10^3\times$ slower than local sampling (Sec. V-C). While this overhead is negligible for OL, it increases FL runtime by nearly $10\times$. Furthermore, the choice of the mechanism significantly impacts performance for both PNoise (**O5**) and CNoise (Tab. III), e.g., with discrete mechanisms being slower than continuous ones.

Privacy-Accuracy Trade-Off. We use plain training (no DP or cryptography) as a baseline with an accuracy of 96.6% on MNIST and 86.0% on Fashion MNIST, while non-DP FL reaches 92.3% and 85.4%, respectively. Introducing DP with CNoise (i.e., non-crypto CNoise) reduces accuracy by up to 3.2 pp from plain training. Smaller ϵ causes further drops (as σ^2 increases), by up to 1.1 pp from $\epsilon = 8$ to $\epsilon = 1$. Cryptography further reduces accuracy due to fixed-point representations and quantization. FL with CNoise shows a more significant accuracy drop (up to 8.7 pp against non-DP FL) compared to OL with CNoise, which remains within 0.5 pp of non-crypto CNoise accuracy. As expected, secure CNoise sampling in FL outperforms LDP, which loses up to 66 pp in accuracy with $\epsilon = 1$. FL with CNoise accuracy is lower than OL since it relies on aggregation of local updates, whereas OL with CNoise can closely match non-DP accuracy due to global dataset updates. This gap increases with heterogeneous (i.e., non-IID) data, e.g., if each client holds only samples of a single class, or classes with conflicting features [99]. Addressing non-IID settings requires specialized optimization methods, e.g., alternating direction multipliers [100]. Accuracy under PNoise is highly sensitive to ϵ and collusion threshold t, since higher t increases the adjusted noise variance (Eq. (3), Sec. V-B). In FL with PNoise (with $n = 100$), half colluding clients ($t = n/2$) cause up to 8.5 pp accuracy reduction compared to CNoise. Here, high collusion ($t = n - 1$) increases the total noise variance above LDP levels. This causes over 70 pp accuracy drop compared to CNoise with $\epsilon = 1$, resulting in even lower accuracy than LDP, i.e., up to 26 pp with $\epsilon = 3$ on FashionMNIST. In contrast, OL typically involves fewer servers than FL clients, reducing the impact of collusion. OL with PNoise, $m = 2$ and high collusion ($t = m - 1$) has a minimal accuracy drop, i.e., at most 1 pp below OL with CNoise. Increasing m to 10 only drops accuracy by $\approx$ 5 pp compared to OL with CNoise [73]. Our extended evaluation (App. O-C) shows that in OL with PNoise and $m = 100$, accuracy drops by $\approx$ 10 pp. Notably, multi-server FL with PNoise could similarly benefit from server-side PNoise, as $m < n$, improving accuracy in case of collusion.

TABLE V: Accuracy on MNIST and FashionMNIST for FL, OL and non-crypto(graphic) DP training, combined with CNoise, PNoise with different collusion thresholds (in parentheses) and LDP. We set $n = 100$ for FL and $m = 2$ for OL.

Accuracy (%)	MNIST			Fashion MNIST		
	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 8$	$\epsilon = 1$	$\epsilon = 3$	$\epsilon = 8$
Non-crypto CNoise	95.8	96.3	96.4	82.8	83.8	83.9
FL with LDP	39.6	63.0	69.6	16.7	57.5	63.7
FL CNoise	87.1	89.7	90.0	78.0	81.6	82.7
FL PNoise ($n/2$)	80.0	87.1	87.7	69.5	79.9	82.3
FL PNoise ($n - 1$)	15.2	47.5	72.7	13.1	31.4	59.2
OL CNoise	95.6	96.2	96.5	82.7	83.6	83.8
OL PNoise ($m - 1$)	95.4	95.8	96.0	82.5	83.5	83.6

A. Designing CPCL Solutions

Drawing from our systematization, Tab. I, Alg. 2 and evaluations above, we find that **(O9) efficient and accurate CPCL solutions require careful design considerations for learning paradigms, cryptographic protocols, and noise sampling techniques.** Enhancing cryptographic CL with DP requires integrating the Perturb phase, Clip and subsampling within GradientCompute. This creates a privacy-utility-performance trade-off space with multiple design choices. Next, we outline key considerations for designing CPCL solutions.

Noise Sampling Strategies. For Perturb, if priorities are performance and ease of implementation, local PNoise sampling and aggregation is the most efficient sampling technique, as it relies solely on cleartext operations. For mechanism choice, Skellam [62] is currently the preferred additive option, as it matches the accuracy of continuous Gaussian using low-precision integers. However, PNoise leads to significant accuracy drops under high collusion ($t = n - 1$), i.e., up to 70 pp in FL (Tab. V). Oblivious PNoise sampling [11], [69] mitigates this, but requires cryptographic protocols, which

introduce performance overhead. If a semi-trusted S', that is oblivious to output, is an option, centralized CNoise sampling introduces minimal overhead and achieves optimal utility (as in CDP) [34]. If strong security is the main requirement and performance is a secondary concern, distributed CNoise sampling is the best choice [74], as it achieves optimal utility without relying on a semi-trusted server.

Cryptographic Constraints. Cryptographic choices constrain noise strategies: techniques with implicit Reveal (e.g., pairwise masking) require the use of PNoise as the aggregated result must already be DP [46]. Single-server solutions (e.g., HE, LWE-based masking [49]) are suited for PNoise or centralized CNoise, while MPC supports all techniques. Furthermore, infrastructure bottlenecks guide design: MPC requires non-colluding servers and is communication-intensive, whereas HE is computation-intensive.

Learning Paradigm Considerations. While FL clients locally run GradientCompute (and Clip), OL requires server-side cryptographic protocols with high overhead (e.g., $200\times$ slowdown with naive Clip, Tab. IV). Notably, the lack of secure clipping optimizations for OL represents a significant gap that hinders performance. Despite FL being the most efficient paradigm (up to $10^4\times$ faster than OL, Tab. IV) it has several drawbacks: it exposes (noisy) gradients to GIA (App. P); may show lower accuracy (e.g., up to 8.5 pp drop vs. OL with CNoise, Tab. V); and requires online clients and cryptographic protocols to handle dropouts (App. I). Conversely, OL approaches with local pre-processing, such as PEA [9], offer a promising middle ground, balancing the strong security of OL with the efficiency of local computations.

VII. Related Works

Only a few works explore the combination of cryptography and DP for CL. Yang et al. [20] evaluate optimizations for DP FL including partial noise aggregation but omit, e.g., OL, multi-server FL and distributed noise sampling. Chatel et al. [101] systematize CL for tree-based models focusing mainly on cryptographic or DP solutions. Other works do not focus on CL. Wagh et al. [33] survey DP and cryptography for distributed analytics, while Fu and Wang [90] benchmark secure sampling protocols for continuous and discrete Laplace and Gaussian. Meisingseth and Rechberger [102] systematize computational DP definitions for multi-party settings. Recent SoKs focus on encrypted CL without DP. Cabrero-Holgueras and Pastrana [16] and Ng and Chow [18] analyze cryptographic training, while Mansouri et al. [17] systematize SecAgg, and Mo et al. [103] systematize training with enclaves. Other works focus only on DP ML. Jayaraman and Evans [38] evaluates utility and privacy of different DP mechanisms. Ponomareva et al. [19] provide guidelines on how to implement DP in centralized and FL settings.

VIII. Research Directions

Building on our systematization, identified gaps (−) and potential enhancements (⊛) from Tab. I, as well as key observations **(O#)**, we propose avenues for future research. Tab. XIII (App. Q) maps these directions to our observations.

(D1) Enhance privacy and performance via pre-processing. Despite its benefits **(O3)**, local pre-processing is underexplored in systematized works. Only PEA [9] clients perform local computations, i.e., feature extraction and global model initialization via local pre-training, improving efficiency without compromising accuracy. Similarly, FL benefits from clients performing many local iterations, e.g., in DP-FedAvg [75], reducing communication overhead. Incorporating public data in pre-processing can enhance feature learning and improve model accuracy while providing strong privacy [104].

(D2) Provide cryptographic building blocks for DP in OL. OL guarantees gradient and model secrecy, preventing GIAs **(O1)**. However, few works explore OL [9], [73], [74], and none provides open-source implementations. Clipping is the main cryptographic bottleneck of OL **(O2)**, yet ML optimizations, e.g. ghost clipping [77], [105] are missing in cryptographic works. Future efforts should focus on efficient, open-source DP building blocks for OL **(O9)**, e.g., sampling, clipping, to foster further research and broader adoption.

(D3) Develop crypto-friendly discrete CNoise sampling for CPCL. Several works [7], [10], [49], [61]–[63] implement discrete PNoise mechanisms e.g., Skellam and discrete Gaussian, to improve utility and performance **(O6)**. However, distributed sampling of discrete CNoise remains unexplored in CPCL, due to challenges like constant runtime **(O8)**, and high overhead [73], [90]. Existing MPC sampling for discrete distributions [55], [90]–[92] do not focus on CL which needs high variance and large number of samples. Their high runtimes make them impractical for CPCL [73], [90] and calls for optimizations of crypto-friendly discrete sampling for CPCL, potentially integrated with quantization-aware training [106].

(D4) Embed DP outside Perturb. To ensure DP, noise addition is not the only avenue, as DP can also be satisfied during quantization or by noise-based cryptography (e.g., LWE) **(O7)**. Future research should explore how to apply DP in other phases For example, randomized rounding satisfies RDP by leveraging DP-focused quantization analysis [107].

(D5) Propose strong user-level DP algorithms for multi-user datasets. Privacy laws like GDPR focus on individuals rather than single records [108]. In distributed settings, where users may contribute multiple records, it is crucial to protect all user data. While for FL standard user-level DP algorithms exist [75], enforcing strong user-level DP when multiple users' data are pooled in a global dataset is challenging **(O4)**. Thus, proposing strong user-level DP algorithms for multi-user datasets is an aspect for future research.

IX. Conclusions

We systematized the landscape of encrypted and differential private collaborative learning (Tab. I), analyzing how to combine cryptography and DP to guarantee input confidentiality and output privacy. Our comprehensive framework (Sec. IV-B) formalized common phases across CPCL paradigms (i.e.,

federated and outsourced learning), identifying noise sampling as the foundational phase. We analyzed noise types, secure sampling techniques, and mechanisms, implementing distributed sampling in LAN and WAN. We evaluated accuracy and runtime overhead of CPCL paradigm across noise sampling techniques, analyzing the performance, accuracy, and privacy trade-offs. Throughout, we identified gaps and possible enhancements in the literature, and highlighted key observations deriving future research directions.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their valuable feedback. This work has received funding from the European Union's Horizon Europe research and innovation program under grant agreement No 101070141 (GLACIATION).

REFERENCES

[1] EU, "General data protection regulation," 2018. [Online]. Available: https://gdpr-info.eu/

[2] D. Evans, V. Kolesnikov, M. Rosulek *et al.*, "A pragmatic introduction to secure multi-party computation," *TTPS*, 2018.

[3] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[4] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," ser. SP. IEEE, 2017.

[5] C. Dwork, "Differential privacy," ser. ICALP, 2006.

[6] A. Bittau, Ú. Erlingsson, P. Maniatis, I. Mironov, A. Raghunathan, D. Lie, M. Rudominer, U. Kode, J. Tinnes, and B. Seefeld, "Prochlo: Strong privacy for analytics in the crowd," ser. SOSP , 2017.

[7] P. Kairouz, Z. Liu, and T. Steinke, "The distributed discrete gaussian mechanism for federated learning with secure aggregation," ser. ICML, 2021.

[8] B. Jayaraman, L. Wang, D. Evans, and Q. Gu, "Distributed learning without distress: Privacy-preserving empirical risk minimization," *NeurIPS*, 2018.

[9] W. Ruan, M. Xu, W. Fang, L. Wang, L. Wang, and W. Han, "Private, efficient, and accurate: Protecting models trained by multi-party learning with differential privacy," ser. SP. IEEE, 2023.

[10] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan, "cpsgd: Communication-efficient and differentially-private distributed sgd," *NeurIPS*, 2018.

[11] D. Byrd, V. Mugunthan, A. Polychroniadou, and T. Balch, "Collusion resistant federated learning with oblivious distributed differential privacy," ser. ICAIF , 2022.

[12] D. Madrigal, A. Manoel, J. Chen, N. Singal, and R. Sim, "Project florida: Federated learning made easy," Microsoft, Tech. Rep., 2023. [Online]. Available: https://www.microsoft.com/en-us/research/publication/project-florida-federated-learning-made-easy/

[13] Z. Xu, Y. Zhang, G. Andrew, C. A. Choquette-Choo, P. Kairouz, H. B. McMahan, J. Rosenstock, and Y. Zhang, "Federated learning of gboard language models with differential privacy," *arXiv preprint arXiv:2305.18465*, 2023.

[14] M. Paulik, M. Seigel, H. Mason, D. Telaar, J. Kluivers, R. van Dalen, C. W. Lau, L. Carlson, F. Granqvist, C. Vandevelde *et al.*, "Federated evaluation and tuning for on-device personalization: System design & applications," *arXiv preprint arXiv:2102.08503*, 2021.

[15] A. D. P. Team, "Learning iconic scenes with differential privacy," 2023. [Online]. Available: https://machinelearning.apple.com/research/scenes-differential-privacy

[16] J. Cabrero-Holgueras and S. Pastrana, "Sok: Privacy-preserving computation techniques for deep learning," *PETS*, 2021.

[17] M. Mansouri, M. Önen, W. B. Jaballah, and M. Conti, "Sok: Secure aggregation based on cryptographic schemes for federated learning," *PETS*, 2023.

[18] L. K. L. Ng and S. S. M. Chow, "SoK: Cryptographic neural-network computation," ser. SP. IEEE, 2023.

[19] N. Ponomareva, H. Hazimeh, A. Kurakin, Z. Xu, C. Denison, H. B. McMahan, S. Vassilvitskii, S. Chien, and A. Thakurta, "How to dp-fy ml: A practical guide to machine learning with differential privacy," *arXiv preprint arXiv:2303.00654*, 2023.

[20] Y. Yang, B. Hui, H. Yuan, N. Gong, and Y. Cao, "{PrivateFL}: Accurate, differentially private federated learning via personalized data transformation," ser. USENIXSec, 2023.

[21] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," ser. CSF. IEEE, 2018.

[22] D. Xu, S. Yuan, and X. Wu, "Achieving differential privacy in vertically partitioned multiparty learning," ser. BigData. IEEE, 2021.

[23] H. Zhu, R. Wang, Y. Jin, and K. Liang, "Pivodl: Privacy-preserving vertical federated learning over distributed labels," *IEEE Transactions on Artificial Intelligence*, 2021.

[24] G.-L. Pereteanu, A. Alansary, and J. Passerat-Palmbach, "Split he: Fast secure inference combining split learning and homomorphic encryption," *arXiv preprint arXiv:2202.13351*, 2022.

[25] T. Khan, K. Nguyen, A. Michalas, and A. Bakas, "Love or hate? share or split? privacy-preserving training using split learning and homomorphic encryption," ser. PST. IEEE, 2023.

[26] H. I. Kanpak, A. Shabbir, E. GenC_c, A. KüpC_cü, and S. Sav, "Cure: Privacy-preserving split learning done right," *arXiv preprint arXiv:2407.08977*, 2024.

[27] Google Scholar, "Security conferences ranking," 2024. [Online]. Available: https://scholar.google.com.ec/citations?view_op=top_venues&hl=en&vq=eng_computersecuritycryptography

[28] J. Zhou, "Security conferences ranking," 2023. [Online]. Available: http://jianying.space/conference-ranking.html

[29] Google Scholar, "Machine learning conferences ranking," 2024. [Online]. Available: https://scholar.google.com.ec/citations?view_op=top_venues&hl=en&vq=eng_artificialintelligence

[30] S. J. Prince, *Understanding Deep Learning*. MIT Press, 2023.

[31] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *FTTCS* , 2014.

[32] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" *SIAM Journal on Computing*, 2011.

[33] S. Wagh, X. He, A. Machanavajjhala, and P. Mittal, "Dp-cryptography: marrying differential privacy and cryptography in emerging applications," *Communications of the ACM*, 2021.

[34] D. Froelicher, J. R. Troncoso-Pastoriza, J. S. Sousa, and J.-P. Hubaux, "Drynx: Decentralized, secure, verifiable system for statistical queries and machine learning on distributed datasets," *IEEE Transactions on Information Forensics and Security*, 2020.

[35] M. Bun and T. Steinke, "Concentrated differential privacy: Simplifications, extensions, and lower bounds," ser. TCC . Springer, 2016.

[36] I. Mironov, "Rényi differential privacy," ser. CSF. IEEE, 2017.

[37] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," ser. CCS, 2016.

[38] B. Jayaraman and D. Evans, "Evaluating differentially private machine learning in practice," ser. USENIXSec, 2019.

[39] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Local privacy and statistical minimax rates," in *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*. IEEE, 2013.

[40] K. Chaudhuri and C. Monteleoni, "Privacy-preserving logistic regression," *NeurIPS*, 2008.

[41] M. Pathak, S. Rane, and B. Raj, "Multiparty differential privacy via aggregation of locally trained classifiers," *NeurIPS*, 2010.

[42] J. Zhang, Z. Zhang, X. Xiao, Y. Yang, and M. Winslett, "Functional mechanism: regression analysis under differential privacy," *arXiv preprint arXiv:1208.0219*, 2012.

[43] N. Papernot, S. Song, I. Mironov, A. Raghunathan, K. Talwar, and Ú. Erlingsson, "Scalable private learning with pate," *arXiv preprint arXiv:1802.08908*, 2018.

[44] B. Ghazi, N. Golowich, R. Kumar, P. Manurangsi, and C. Zhang, "Deep learning with label differential privacy," *NeurIPS*, 2021.

[45] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," ser. GlobalSIP . IEEE, 2013.

[46] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," ser. SIGSAC , 2017.

[47] W. Diffie and M. E. Hellman, "New directions in cryptography," in *Democratizing Cryptography: The Work of Whitfield Diffie and Martin Hellman*, 2022.

[48] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *JACM*, 2009.

[49] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near, "Efficient differentially private secure aggregation for federated learning via hardness of learning with errors," ser. USENIXSec, 2022.

[50] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," ser. EUROCRYPT, 1999.

[51] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," ser. ASIACRYPT, 2017.

[52] I. Damgård and M. Jurik, "A generalisation, a simplification and some applications of paillier's probabilistic public-key system," ser. PKC , 2001.

[53] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," ser. NDSS, 2011.

[54] A. C.-C. Yao, "How to generate and exchange secrets," in *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 1986.

[55] H. Keller, H. Möllering, T. Schneider, O. Tkachenko, and L. Zhao, "Secure noise sampling for dp in mpc with finite precision," ser. ARES , 2024.

[56] F. Tramèr, R. Shokri, A. San Joaquin, H. Le, M. Jagielski, S. Hong, and N. Carlini, "Truth serum: Poisoning machine learning models to reveal their secrets," ser. SIGSAC . ACM, 2022.

[57] M. Aerni, J. Zhang, and F. Tramèr, "Evaluations of machine learning privacy defenses are misleading," *arXiv preprint arXiv:2404.17399*, 2024.

[58] R. Wu, X. Chen, C. Guo, and K. Q. Weinberger, "Learning to invert: Simple adaptive attacks for gradient inversion in federated learning," ser. UAI . PMLR, 2023.

[59] A. G. Sébert, M. Checri, O. Stan, R. Sirdey, and C. Gouy-Pailler, "Combining homomorphic encryption and differential privacy in federated learning," ser. PST . IEEE, 2023.

[60] Z. Bu, J. Dong, Q. Long, and W. J. Su, "Deep learning with gaussian differential privacy," *Harvard data science review*, 2020.

[61] L. Wang, R. Jia, and D. Song, "D2p-fed: Differentially private federated learning with efficient communication," *arXiv preprint arXiv:2006.13039*, 2020.

[62] N. Agarwal, P. Kairouz, and Z. Liu, "The skellam mechanism for differentially private federated learning," *NeurIPS*, 2021.

[63] E. Bao, Y. Zhu, X. Xiao, Y. Yang, B. C. Ooi, B. H. M. Tan, and K. M. M. Aung, "Skellam mixture mechanism: a novel approach to federated learning with differential privacy," *Proceedings of the VLDB Endowment*, 2022.

[64] W.-N. Chen, A. Ozgur, and P. Kairouz, "The poisson binomial mechanism for unbiased federated learning with secure aggregation," ser. ICML, 2022.

[65] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," ser. AISec , 2019.

[66] L. Lyu, "Lightweight crypto-assisted distributed differential privacy for privacy-preserving distributed learning," ser. IJCNN . IEEE, 2020.

[67] X. Gu, M. Li, and L. Xiong, "Precad: Privacy-preserving and robust federated learning via crypto-aided differential privacy," *arXiv preprint arXiv:2110.11578*, 2021.

[68] Y. Allouah, R. Guerraoui, and J. Stephan, "Towards trustworthy federated learning with untrusted participants," *arXiv preprint arXiv:2505.01874*, 2025.

[69] V. Mugunthan, A. Polychroniadou, D. Byrd, and T. H. Balch, "Smpai: Secure multi-party computation for federated learning," in *NeurIPS Workshop on Robust AI in Financial Services*, 2019.

[70] V. Bindschaedler, S. Rane, A. E. Brito, V. Rao, and E. Uzun, "Achieving differential privacy in secure multiparty data aggregation protocols on star networks," ser. CODASPY . ACM, 2017.

[71] M. Chase, R. Gilad-Bachrach, K. Laine, K. Lauter, and P. Rindal, "Private collaborative neural network learning," Cryptology ePrint Archive, Paper 2017/762, 2017.

[72] K. Iwahana, N. Yanai, J. P. Cruz, and T. Fujiwara, "Spgc: Integration of secure multiparty computation and differential privacy for gradient computation on collaborative learning," *JIP* , 2022.

[73] S. Das, S. R. Chowdhury, N. Chandran, D. Gupta, S. Lokam, and R. Sharma, "Communication efficient secure and private multi-party deep learning," *PETS*, 2025.

[74] S. Pentyala, D. Railsback, R. Maia, R. Dowsley, D. Melanson, A. Nascimento, and M. De Cock, "Training differentially private models with secure multiparty computation," *arXiv preprint arXiv:2202.02625*, 2022.

[75] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.

[76] Y.-X. Wang, B. Balle, and S. P. Kasiviswanathan, "Subsampled rényi differential privacy and analytical moments accountant," ser. AISTATS , 2019.

[77] X. Li, F. Tramer, P. Liang, and T. Hashimoto, "Large language models can be strong differentially private learners," *arXiv preprint arXiv:2110.05679*, 2021.

[78] J. Ye and R. Shokri, "Differentially private learning needs hidden state (or much faster convergence)," *arXiv preprint arXiv:2203.05363*, 2022.

[79] S. Panda, "Polynomial approximation of inverse sqrt function for fhe," in *International Symposium on Cyber Security, Cryptology, and Machine Learning*, ser. CSCML, 2022.

[80] W.-j. Lu, Y. Fang, Z. Huang, C. Hong, C. Chen, H. Qu, Y. Zhou, and K. Ren, "Faster secure multiparty computation of adaptive gradient descent," in *Workshop on Privacy-Preserving Machine Learning in Practice*, 2020.

[81] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "Cryptgpu: Fast privacy-preserving machine learning on the gpu," ser. SP. IEEE, 2021.

[82] P. Kairouz, B. McMahan, S. Song, O. Thakkar, A. Thakurta, and Z. Xu, "Practical and private (deep) learning without sampling or shuffling," ser. ICML, 2021.

[83] L. Chua, B. Ghazi, Y. Huang, P. Kamath, R. Kumar, D. Liu, P. Manurangsi, A. Sinha, and C. Zhang, "Mind the privacy unit! user-level differential privacy for language model fine-tuning," ser. COLM, 2024.

[84] C. A. Choquette-Choo, N. Dullerud, A. Dziedzic, Y. Zhang, S. Jha, N. Papernot, and X. Wang, "Capc learning: Confidential and private collaborative learning," ser. ICLR , 2021.

[85] C. A. Neff, "Verifiable mixing (shuffling) of elgamal pairs," *VHTi Technical Document, VoteHere, Inc*, 2003.

[86] "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. Available: https://www.tensorflow.org/

[87] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *NeurIPS*, 2019.

[88] A. M. Law, W. D. Kelton, and W. D. Kelton, *Simulation modeling and analysis*. Mcgraw-hill New York, 2007.

[89] G. E. Box and M. E. Muller, "A note on the generation of random normal deviates," *The annals of mathematical statistics*, 1958.

[90] Y. Fu and T. Wang, "Benchmarking secure sampling protocols for differential privacy," ser. CCS, 2024.

[91] C. Wei, R. Yu, Y. Fan, W. Chen, and T. Wang, "Securely sampling discrete gaussian noise for multi-party differential privacy," ser. CCS, 2023.

[92] F. Meisingseth, C. Rechberger, and F. Schmid, "Practical two-party computational differential privacy with active security," *PETS*, 2025.

[93] C. L. Canonne, G. Kamath, and T. Steinke, "The discrete gaussian for differential privacy," *NeurIPS*, 2020.

[94] D. E. Knuth, *The art of computer programming*. Pearson Education, 1997.

[95] M. Keller, "Mp-spdz: A versatile framework for multi-party computation," Cryptology ePrint Archive, Paper 2020/521, 2020.

[96] M. Keller, E. Orsini, and P. Scholl, "Mascot: faster malicious arithmetic secure computation with oblivious transfer," ser. SIGSAC . ACM, 2016.

[97] F. Granqvist, C. Song, Á. Cahill, R. van Dalen, M. Pelikan, Y. S. Chan, X. Feng, N. Krishnaswami, V. Jina, and M. Chitnis, "pfl-research: simulation framework for accelerating research in private federated learning," *arXiv preprint arXiv:2404.06430*, 2024.

[98] B. Knott, S. Venkataraman, A. Hannun, S. Sengupta, M. Ibrahim, and L. van der Maaten, "Crypten: Secure multi-party computation meets machine learning," 2021.

[99] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *FTML*, 2021.

[100] Y. Miao, D. Kuang, X. Li, S. Xu, H. Li, K.-K. R. Choo, and R. H. Deng, "Privacy-preserving asynchronous federated learning under non-iid settings," *IEEE Transactions on Information Forensics and Security*, 2024.

[101] S. Chatel, A. Pyrgelis, J. R. Troncoso-Pastoriza, and J.-P. Hubaux, "Sok: Privacy-preserving collaborative tree-based model learning," *PETS*, 2021.

[102] F. Meisingseth and C. Rechberger, "Sok: Computational and distributed differential privacy for mpc," *Cryptology ePrint Archive*, 2024.

[103] F. Mo, Z. Tarkhani, and H. Haddadi, "Machine learning with confidential computing: A systematization of knowledge," *arXiv preprint arXiv:2208.10134*, 2022.

[104] F. Tramer and D. Boneh, "Differentially private learning needs better features (or much more data)," *arXiv preprint arXiv:2011.11660*, 2020.

[105] Y. Ding, X. Wu, H. Wang, and W. Pan, "Dpformer: Learning differentially private transformer on long-tailed data," *arXiv preprint arXiv:2305.17633*, 2023.

[106] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," ser. CVPR . IEEE, 2018.

[107] Y. Youn, Z. Hu, J. Ziani, and J. Abernethy, "Randomized quantization is all you need for differential privacy in federated learning," *arXiv preprint arXiv:2306.11913*, 2023.

[108] A. Salem, G. Cherubin, D. Evans, B. Köpf, A. Paverd, A. Suri, S. Tople, and S. Zanella-Béguelin, "Sok: Let the privacy games begin! a unified treatment of data inference privacy in machine learning," ser. SP. IEEE, 2023.

[109] D. Desfontaines, "Averaging risk: Rényi dp & zero-concentrated dp," 2022, ted is writing things (personal blog). [Online]. Available: https://desfontain.es/privacy/renyi-dp-zero-concentrated-dp.html

[110] B. Balle, G. Barthe, and M. Gaboardi, "Privacy amplification by subsampling: Tight analyses via couplings and divergences," *NeurIPS*, 2018.

[111] S. Laur, J. Willemson, and B. Zhang, "Round-efficient oblivious database manipulation," ser. ISC , 2011.

[112] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, "Secure single-server aggregation with (poly) logarithmic overhead," ser. SIGSAC . ACM, 2020.

[113] M. O. Rabin, "How to exchange secrets with oblivious transfer," *Cryptology ePrint Archive*, 2005.

[114] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "Acorn: Input validation for secure aggregation," *Cryptology ePrint Archive*, 2022.

[115] M. Franklin and M. Yung, "Communication complexity of secure computation," ser. STOC. ACM, 1992.

[116] J. Camenisch, R. Chaabouni, and A. Shelat, "Efficient protocols for set membership and range proofs," ser. ASIACRYPT, 2008.

[117] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Secure linear regression on vertically partitioned datasets." *IACR Cryptol. ePrint Arch.*, 2016.

[118] Z. Wang, G. Yang, H. Dai, and C. Rong, "Privacy-preserving split learning for large-scaled vision pre-training," *IEEE Transactions on Information Forensics and Security*, 2023.

[119] C. Thapa, P. C. M. Arachchige, S. Camtepe, and L. Sun, "Splitfed: When federated learning meets split learning," in *AAAI Conference on Artificial Intelligence*, 2022.

[120] Y. Liu, Y. Kang, T. Zou, Y. Pu, Y. He, X. Ye, Y. Ouyang, Y.-Q. Zhang, and Q. Yang, "Vertical federated learning: Concepts, advances, and challenges," *IEEE transactions on knowledge and data engineering*, 2024.

[121] F. Boemer, Y. Lao, R. Cammarota, and C. Wierzynski, "Ngraph-he: A graph compiler for deep learning on homomorphically encrypted data," ser. CF . ACM, 2019.

[122] D. Demmler, T. Schneider, and M. Zohner, "Aby-a framework for efficient mixed-protocol secure two-party computation." in *NDSS*, 2015.

[123] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[124] A. T. Suresh, X. Y. Felix, S. Kumar, and H. B. McMahan, "Distributed mean estimation with limited communication," ser. ICML, 2017.

[125] R. Taiello, M. Önen, C. Gritti, and M. Lorenzi, "Let them drop: Scalable and efficient federated learning solutions agnostic to stragglers," ser. ARES , 2024.

[126] F. KarakoC_c, A. KüpC_cü, and M. Önen, "Fault tolerant and malicious secure federated learning," in *International Conference on Cryptology and Network Security*, 2024.

[127] M. Mansouri, M. Önen, and W. Ben Jaballah, "Learning from failures: Secure and fault-tolerant aggregation for federated learning," ser. ACSAC, 2022.

[128] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, "Our data, ourselves: Privacy via distributed noise generation," ser. EUROCRYPT. Springer, 2006.

[129] Y. Lyubarskii and R. Vershynin, "Uncertainty principles and vector quantization," *IEEE Transactions on Information Theory*, 2010.

[130] C. Sabater, F. Hahn, A. Peter, and J. Ramon, "Private sampling with identifiable cheaters," *PETS*, 2022.

[131] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[132] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[133] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," ser. AISTATS , 2017.

[134] S. Caldas, S. M. K. Duddu, P. Wu, T. Li, J. Konečnỳ, H. B. McMahan, V. Smith, and A. Talwalkar, "Leaf: A benchmark for federated settings," *arXiv preprint arXiv:1812.01097*, 2018.

[135] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, "Emnist: Extending mnist to handwritten letters," ser. IJCNN . IEEE, 2017.

[136] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," ser. SP. IEEE, 2019.

[137] Y. Zhang, G. Bai, M. A. P. Chamikara, M. Ma, L. Shen, J. Wang, S. Nepal, M. Xue, L. Wang, and J. Liu, "Agrevader: Poisoning membership inference against byzantine-robust federated learning," ser. THEWEBCONF . ACM.

[138] Y. Wen, L. Marchyok, S. Hong, J. Geiping, T. Goldstein, and N. Carlini, "Privacy backdoors: Enhancing membership inference through poisoning pre-trained models," *arXiv preprint arXiv:2404.01231*, 2024.

[139] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," ser. SP. IEEE, 2022.

[140] Y. Chen, C. Shen, Y. Shen, C. Wang, and Y. Zhang, "Amplifying membership exposure via data poisoning," *NeurIPS*, 2022.

[141] S. Feng and F. Tramèr, "Privacy backdoors: Stealing data with corrupted pretrained models," *arXiv preprint arXiv:2404.00473*, 2024.

[142] H. Chaudhari, M. Jagielski, and A. Oprea, "Safenet: Mitigating data poisoning attacks on private machine learning." *IACR Cryptol. ePrint Arch.*, 2022.

[143] J. Jin, E. McMurtry, B. I. Rubinstein, and O. Ohrimenko, "Are we there yet? timing and floating-point attacks on differential privacy systems," ser. SP. IEEE, 2022.

[144] I. Mironov, "On significance of the least significant bits for differential privacy," ser. CCS. ACM, 2012.

[145] D. Desfontaines, "Tiny bits matter: precision-based attacks on differential privacy," 2022. [Online]. Available: https://www.tmlt.io/resources/tiny-bits-matter-precision-based-attacks/-on-differential-privacy

[146] J. C. Zhao, A. Sharma, A. R. Elkordy, Y. H. Ezzeldin, S. Avestimehr, and S. Bagchi, "Loki: Large-scale data reconstruction attack against federated learning through model manipulation," ser. SP. IEEE, 2023.

[147] B. Jayaraman and D. Evans, "Are attribute inference attacks just imputation?" 2022. [Online]. Available: https://arxiv.org/abs/2209.01292

[148] C. Song and V. Shmatikov, "Overlearning reveals sensitive attributes," *arXiv preprint arXiv:1905.11742*, 2019.

[149] Y. Liu, R. Wen, X. He, A. Salem, Z. Zhang, M. Backes, E. De Cristofaro, M. Fritz, and Y. Zhang, "{ML-Doctor}: Holistic risk assessment of inference attacks against machine learning models," ser. USENIXSec, 2022.

[150] M. Fredrikson, E. Lantz, S. Jha, S. Lin, D. Page, and T. Ristenpart, "Privacy in pharmacogenetics: An {End-to-End} case study of personalized warfarin dosing," ser. USENIXSec, 2014.

APPENDIX A
SYSTEMATIC SEARCH

Here we detail the methodology for our preliminary and systematic search, and the selection of relevant works. As a starting point, we considered top-tier conferences and journals in security and ML, based on empirical rankings [27]–[29]. Those rankings measure either the h-5 index, i.e., the largest number h such that h papers have at least h citations in the last 5 years, or the Conference Impact Factor defined in [28]. For security, we considered the joint set of the top-5 venues from [27] and [28], including cryptography conferences: IEEE Symposium on Security and Privacy, USENIX Security Symposium, ACM Conference on Computer and Communications Security, Network and Distributed System Security Symposium, IEEE Transactions on Information Forensics and Security (Journal), Computer & Security (Journal), International Cryptology Conference, Cryptology-EUROCRYPT. For ML, we selected top-5 venues from [29]: Neural Information Processing Systems, International Conference on Learning Representations, International Conference on Machine Learning, AAAI Conference on Artificial Intelligence, IEEE Transactions on Systems. The query for Google Scholar and BASE was: distributed OR collaborative "multi-party" OR multiparty OR homomorphic OR "privacy-preserving" OR secure "differential-privacy" OR "differential privacy" learning OR training source:"Conference Name". Here "Conference Name" is replaced by each of the 13 venues listed above. Notably, our full selection of works, i.e., including also cited by or citing found works, includes relevant papers beyond these initial venues.

APPENDIX B
SUMMARY OF NOTATION

Tab. VI provides a summary of notation and symbols with references to sections which introduce those symbols.

APPENDIX C
RELAXED DP DEFINITIONS

Dwork et al. [31] introduced *advanced composition* to offer tighter privacy bounds, but the composition bound for (ϵ, δ)-DP. However, the composition bound is still loose, requiring relaxed DP definitions. Next, we formalize Rényi DP, zero-concentrated DP and their conversion lemmas to (ϵ, δ)-DP. Table VII reports the conversion formulas and compares the privacy guarantee achieved after k-fold composition.

Definition 1: (Rényi Differential Privacy) [36] RDP is a generalization of the notion of differential privacy based on the Rényi divergence:

$$D_\alpha(P||Q) \triangleq \frac{1}{\alpha - 1} \ln E_{x\sim Q}\left(\frac{P(x)}{Q(x)}\right)^\alpha,$$

for two probability distributions P and Q. A randomized mechanism $\mathcal{M}$ satisfies (α, ϵ)-RDP if for any neighboring dataset D_1, D_2 it holds:

$$D_\alpha(\mathcal{M}(D_1)||\mathcal{M}(D_2)) \leq \epsilon. \tag{4}$$

TABLE VI: Summary of Notation and Symbols.

Symbol	Description
Learning Paradigms	
CL	Collaborative learning (Sec. I).
CPCL	Cryptographic and differentially private collaborative learning (Sec. I).
FL	Federated learning (Sec. IV-C).
OL	Outsourced learning (Sec. IV-D).
Parties & Sets (Sec. III-B)	
$\mathcal{C}$	Set of clients which are input and output parties (optionally can be also computation parties).
$\mathcal{S}$	Set of servers which are computation parties.
$\mathcal{O}$	Set of output parties, which includes clients and optionally servers, i.e., $\mathcal{C} \subseteq \mathcal{O} \subseteq \mathcal{C} \cup \mathcal{S}$.
C_i, S_j	The i-th client and j-th server.
S'	Semi-trusted server with no access to computation outputs.
n	Number of clients.
m	Number of servers.
t	Collusion threshold (maximum number of colluding parties).
Data & Learning (Sec. III-A)	
D, D_i	Global dataset and local dataset of client C_i.
$\theta, \theta^{(k)}$	Model parameters, parameters at step k.
$g, \mathbf{G}$	Gradient vector and Set of gradients.
$\tilde{g}$	Noisy gradient $(g + \psi)$.
K	Gradient clipping threshold.
η	Learning rate.
B, b	Batch, mini-batch size.
Differential Privacy & Noise Sampling	
(ϵ, δ)	DP budget parameters (Sec. III-A).
ψ	Noise sample drawn from a distribution in Tab. II (Sec. III-A).
PerturbGradient	Noise added to gradient updates (Sec. III-A).
PerturbOutput	Noise added to trained parameters (Sec. III-A).
PNoise	Partial, non-DP noise sampled locally. The aggregation of PNoise satisfies DP (Sec. IV).
CNoise	Centralized noise term sampled globally which satisfies CDP (Sec. IV).
τ	Tuple of noise parameters which defines noise type (PNoise/CNoise), noise mechanism (Tab. II, Sec. V) and distribution parameters (e.g., σ^2 for (ϵ, δ)-DP) (Sec. IV-B).
Cryptography & Formatting (Sec. III-B)	
$[x]$ or $[x]_s$	Encrypted or secret-shared value of x.
Phase	Computation on global and joint encrypted data (e.g., via MPC).
Perturb	Centralized sampling via a semi-trusted server S'.

TABLE VII: Comparison of different relaxed DP definitions from Jayaraman and Evans [38, Tab. 1]

	Advanced composition (ϵ, δ)-DP	ρ-zCDP	(α, ϵ)-RDP
Conversion to (ϵ, δ)-DP	-	$(\rho + 2\sqrt{\rho \log \frac{1}{\delta}}, \delta)$-DP	$(\epsilon + \log \frac{1}{\delta}/(\alpha - 1), \delta)$-DP
Composition of k ϵ-DP mech.	$(\epsilon\sqrt{2k \log \frac{1}{\delta}} + k\epsilon(e^{\epsilon} - 1), \delta)$-DP	$(\epsilon\sqrt{2k \log \frac{1}{\delta}} + k\epsilon^2/2, \delta)$-DP	$(4\epsilon\sqrt{2k \log \frac{1}{\delta}}, \delta)$-DP

Lemma 1: (Conversion from RDP to (ϵ, δ)-DP) [36] The RDP definition can be easily converted to (ϵ, δ)-DP when $0 < \delta < 1$. If a randomized mechanism $\mathcal{M}$ holds for (α, ϵ)-RDP then $\mathcal{M}$ is $(\epsilon + \log(1/\delta)/(\alpha - 1), \delta)$-DP.

Definition 2: (zero-Concentrated Differential Privacy) [35] A randomized mechanism $\mathcal{M}$ satisfies (ξ, ρ)-zCDP if for any neighboring dataset D_1, D_2 and for all $\alpha \in (1, \infty)$:

$$D_\alpha(\mathcal{M}(D_1)||\mathcal{M}(D_2)) \leq \xi + \rho\alpha, \tag{5}$$

where $D_\alpha(\cdot)$ is the α-Rényi divergence. Bun et al. [35] define $(0, \rho)$-zCDP as ρ-zCDP.

Lemma 2: (Conversion from ρ-zCDP to (ϵ, δ)-DP) [35] If a randomized mechanism M provides ρ-zCDP, then M is $(\rho + 2\sqrt{\rho \log(1/\delta)}, \delta)$-DP for any $\delta > 0$

Both Rényi DP and (zero) concentrated DP uses the Rényi divergence to quantify the privacy loss. The Rényi divergence carries a parameter α, which define the moments of the divergence. The main difference between RDP and zCDP lies in the fact that RDP applies one moment at a time, whereas zCDP requires a linear bound on all positive moments of a privacy loss variable [36]. Having the privacy loss value linked to one moment allows for more accurate numerical analysis [36]. Notably, while (ϵ, δ)-DP defines the worst-case privacy loss, which can be infinite with a non-zero probability δ [7], [36], zCDP and RDP do not allow for such privacy breaches as they are defined on the average privacy loss [109].

Ponomareva et al. [19, Tab. 2] show the evolution of DP-training bounds with T iteration of DP-SGD, with a sampling parameter q. Using moments accountant on RDP and converting in (ϵ, δ)-DP, the bound on ϵ decreases by a factor of $\sqrt{T}$, whereas the bound on δ reduces by qT.

A. Local DP

To avoid relying on a trusted third party as in CDP (Def. 1), in *local* DP (LDP) [32], each data owner applies the perturbation independently.

Definition 3: Formally, $\mathcal{M}$ satisfies (ϵ, δ)-local DP if:

$$Pr[\mathcal{M}(x_1) \in \mathcal{S}] \leq \exp(\epsilon) \times Pr[\mathcal{M}(x_2) \in \mathcal{S}] + \delta \tag{6}$$

for any pair of values $x_1, x_2 \in D$ and all $\mathcal{S} \subseteq Range(\mathcal{M})$.

B. Computational DP

While CDP and LDP consider an unbounded adversary, cryptography bounds the adversary's capabilities.

Computational DP [31] adapts Def. 1 to a bounded polynomial-time adversary modelling the probability to break a cryptographic scheme.

Definition 4: A randomized algorithm $\mathcal{C}_\kappa : \mathcal{X}_n \to Y$ is *ϵ-computationally differentially private* if and only if for all the neighboring datasets (D_1, D_2) and for all nonuniform polynomial (in κ) algorithms T,

$$Pr[T(\mathcal{C}_\kappa(D_1)) = 1] \leq \exp(\epsilon) \times Pr[T(\mathcal{C}_\kappa(D_2)) = 1] + \delta(\kappa). \tag{7}$$

Here, $\delta(\cdot) = \delta + \text{negl}(\cdot)$, where $\text{negl}(\cdot)$ is any function that grows slower than the inverse of any polynomial. The algorithm $\mathcal{C}_\kappa$ runs in time polynomial in n, $\log|X|$, and κ. The term $\text{negl}(\kappa)$ represents the probability of violating a cryptographic scheme, e.g., guessing the decryption key [31].

APPENDIX D
PRIVACY AMPLIFICATION

Randomly sampling a subset of the data for each iteration amplifies the privacy guarantee of DP-SGD. The uncertainty of a sample being selected in a training iteration makes it harder for an adversary to infer the presence of a specific record in the dataset. Subsampling amplification allows DP-SGD to achieve strong privacy-utility trade-offs, as it can reduce the noise variance for the same privacy guarantee [37]. For each iteration B samples are selected out of N, i.e., each with a probability $q = B/N$. Analytically the amplification is $(O(q(\exp(\epsilon) - 1)), O(q\delta))$-DP [19]. For small values of ϵ ($\epsilon \leq 1$), the amplification is of a factor of q; this factor decreases as ϵ increases. The batch size B is crucial, as larger B values reduce the noise variance but require more gradient computations, and smaller B values increase the noise variance but reduce the computation overhead. According to [37], the best batch size B is roughly $\sqrt{N}$.

Subsampling Techniques. The subsampling amplification requires true sampling and not shuffling [19]. The two main subsampling techniques are *Poisson* [76] and *Uniform* [110]. The two techniques rely on a different notion of neighboring datasets in Def. 1, i.e., *add/remove* or *replace*. The add/remove notion is used for Poisson subsampling and consists of adding or removing a sample from a dataset D_1 to obtain the neighboring one D_2. This leads to a variable dataset size that translates into a variable batch size for each training iteration with the Poisson subsampling. The replace notion is used for the uniform subsampling and consists of replacing a sample in D_1 with another to obtain D_2. This leads to a fixed batch size for each training iteration with uniform subsampling [19].

Subsampling in FL and OL. In FL, the subsampling amplification happens on clients, i.e., for each iteration the servers select a subset of clients. This approach is particularly challenging in federated settings where the availability of the clients can be different at each round. To solve this issue Kairouz et al. [82] propose a DP variant of the follow-the-regularized-leader (FTRL) algorithm, i.e., DP-FTRL, that does not rely on privacy amplification, but on a tree aggregation trick. In OL, the subsampling happens on the global joint

dataset and needs to be implemented in MPC, i.e., the servers need to collaboratively sample Poisson random variable with rate q and selected the indexes of the samples to be used in the training iteration. From the analyzed works, PEA [9] implements a resharing-based oblivious shuffling protocol [111], but this approach does not satisfy either Poisson or Uniform amplification [19]. To perform subsampling in our evaluation of DP overhead in OL, reported in Tab. IV (Sec. VI), we implement the *Poisson* subsampling by letting one server to perform the subsampling locally and then reshare the indexes to the other servers. This approach is efficient, i.e., the sampling requires less than 0.1% of the GradientCompute time, but leaks the indexes of the selected samples which are secret shared among the servers. An alternative solutions could be to let each client subsample its own dataset locally and secret share the subsampled data for each iteration, but this would require additional communication rounds.

APPENDIX E
CRYPTOGRAPHIC TECHNIQUES

Next, we expand on cryptographic techniques briefly introduced in Sec. III-B. In particular, we provide more details for garbled circuits and learning with errors, and we discuss zero-knowledge proofs which can be used as mitigation for privacy attacks discussed in App. P. Additionally, we provide a complexity analysis of the core phases of cryptographic techniques summarized in Tab. VIII.

A. Garbled Circuits

Garbled circuits (GC) [54] is a two-party protocol to securely evaluate a function f represented as a Boolean circuit C. One party (garbler), runs Garble, creating a GC whose truth table entries consists of random labels. Another party (evaluator), runs Eval on the GC, evaluating the circuit. To let the evaluator only learn random labels corresponding to its input while hiding the evaluator's input from the garbler, they run a cryptographic protocol called *oblivious transfer* [113]. GC requires *correctness*, i.e., outputs of garbled C and f match, and *secrecy*, i.e., Eval reveals only the GC output.

B. Learning with Errors

Learning with Errors (LWE) [48] can be seen as the problem of decoding from a random linear code. The LWE problem is formalized as finding an unknown vector $s \in \mathbb{Z}_q^n$ such that $b = As + e \in \mathbb{Z}_q^m$. Where $A \in \mathbb{Z}_q^{m \times n}$ is a matrix built from m random vectors $a_i \in \mathbb{Z}_q^n$, and $e \in \mathbb{Z}_q^m$ is the error vector. The challenge of LWE comes from the e entries, which are sampled from a suitable probability distribution on $\mathbb{Z}_q$ disrupting the linear relation among the equations. Notably, cryptographic schemes based on LWE are additively homomorphic for both the key and the message, i.e., given two messages (m_1, m_2), an encryption algorithm Enc and two keys (s_1, s_2), $Enc(s_1, m_1) + Enc(s_2, m_2) = Enc(s_1 + s_2, m_1 + m_2)$ [114].

LWE in FL. In Stevens et al. [49], each client C_i samples its secret s_i and the error vector e_i from a discrete Gaussian distribution. C_i masks the local update v_i with b_i, resulting in $h_i = v_i + b_i$ where $b_i = As_i + e_i$, and sends h_i to the server S. S computes $h_{sum} = b_{sum} + v_{sum}$ where $b_{sum} = As_{sum} + e_{sum}$. To remove the term As_{sum} and retrieve the noisy global update $v_{sum} + e_{sum}$, each client secret shares, via *packed Shamir secret sharing* [115], its s_i to all the clients[2]. A set of t clients reconstruct s_{sum} and send the value to S.

C. Zero-Knowledge Proofs

A zero knowledge proof (ZKP) allows a prover to convince a verifier that it knows a secret x satisfying a public predicate $C(x) = 1$, without revealing x itself [2]. Maliciously secure protocols use ZKPs to prove the validity of computations over private inputs without exposing those inputs. For example, Drynx [34] leverages ZKPs in FL to verify both local updates and their aggregation. Specifically, clients provide range proofs [116] to ensure their local model updates fall within valid bounds, while servers generate ZKPs to prove the correctness of the aggregation process.

D. Complexity Analysis of Cryptographic Techniques

We now detail how specific cryptographic techniques implement the core phases, highlighting the trade-offs in communication and computation for non-optimized version of protocols summarized in Tab. VIII.

Setup. Cryptographic techniques have different Setup requirements. MPC requires multiple non-colluding servers but no key generation for clients, since Shr uses local randomness. However, MPC typically requires servers to exchange cryptographic material in the offline phase. In contrast, HE and masking operate with a single server and require a secret key known by (or distributed to) clients. In HE, clients agree on a set of keys (pk, sk). Key generation can be done via a trusted dealer in $O(n)$ messages sent to clients [59], or via distributed key generation requiring clients interaction in $O(n^2)$ messages [65], [70]. Masking requires clients to exchange pairwise seeds for mask generation, in $O(n^2)$ messages. Communication can be reduced to $O(n \log n)$ via k-regular graphs, i.e., exchanging seeds with $k = \log n$ neighbors [112], or via LWE-based masking [48] where matrix A is publicly shared ($O(1)$ messages).

Protect. In MPC, each client secret shares (Shr) data among m servers, in $O(m)$ messages. Instead, in single-server settings, clients leverage HE or masking to encrypt (Enc) or mask the data sending $O(1)$ messages to the server.

Reveal. MPC typically incurs $O(mn)$ total messages if clients reconstruct (Rec) the result by receiving $O(m)$ messages from servers (e.g., summing shares in additive SS), or $O(m^2 + n)$ if servers reconstruct and send results to clients. With HE, clients can use a common secret key for decryption (Dec), risking that a client and the server can collude to decrypt other clients' data [59]. A safer option is distributed decryption via threshold HE, where clients communicate directly in $O(n^2)$ messages [59] or the servers

[2]Bell et al. [114] stated that the reconstruction protocol for the secret s could reveal some information about s, breaking confidentiality.

TABLE VIII: Total comp(utation) and comm(unication) complexity of cryptographic techniques across phases (for non-optimized implementations).

Technique	Setup (by $\mathcal{C}$)		Protect (by $\mathcal{C}$)		Reveal (by either $\mathcal{C}$ or $\mathcal{S}$)	
	Comp	Comm	Comp	Comm	Comp	Comm
MPC	$O(1)$	$O(1)$	$O(nm)$	$O(nm)$	$O(m)$	$O(nm)$ if Rec by $\mathcal{C}$ $O(m^2+n)$ if Rec by $\mathcal{S}$
HE	$O(n)$	$O(n^2)$ if distributed key gen [70] $O(n)$ via trusted dealer [59]	$O(n)$	$O(n)$	$O(1)$	$O(n^2)$ if distributed decryption by $\mathcal{C}$ [59] $O(n)$ if server act as relay (Dec by $\mathcal{C}$ or $\mathcal{S}$) [65]
Masking	$O(n)$	$O(n^2)$ if pair-wise masking [46] $O(n \log n)$ via k-regular graphs [112] $O(1)$ if LWE-based [49]	$O(n)$	$O(n)$	$O(1)$ [10]	$O(1)$ if pair-wise masking [10] $O(n)$ if dropout resistant [46] $O(n)$ if LWE-based [49]

act as a relay ($O(n)$ messages) [65], [70]. For pair-wise masking Reveal is implicit as masks cancel out in Aggregate ($O(1)$ messages). In LWE-based masking, however, clients reconstruct the aggregated $\sum_i^n s_i$ in $O(n)$ messages, with a new key per epoch.

Appendix F
Framework Generality

Figure 2 illustrates the generalization of our CPCL framework across different collaborative learning paradigms as introduced in Sec. IV-B. We detail the execution flows for FL (including both client- and server-side noise sampling) and OL, as formalized in Alg. 2. Furthermore, we demonstrate the framework's modular extensibility to peer-to-peer (P2P), split learning (SL) [24]–[26], and vertically partitioned data in FL (VFL) [22], [23] and OL (VOL) [117].

Specifically, in *peer-to-peer* learning clients act as servers ($\mathcal{C} = \mathcal{S}$), throughout the training. In *split learning* [118], [119] the model is partitioned across clients and servers, which perform GradientCompute on their respective partitions. Here, clients apply Protect on intermediate activations and send them to servers. Aggregate operates at layer level rather than on full-model gradients, and gradients are not combined across clients. Hence, each client can execute Perturb locally adding CNoise to its gradients. Meanwhile, for model partitions trained by servers, Perturb can either sample PNoise locally or CNoise via MPC or a semi-trusted party. In *vertical federated learning* [120] clients hold different features for the same samples. Thus, Setup also includes private entity alignment to link records across datasets [120]. Clients train their local model and execute Protect on intermediate activations. Servers perform Aggregate on local activations, and then execute GradientCompute on the global loss. The gradients are then sent back to clients. As in split learning, Aggregate is executed on intermediate results and the same considerations on Perturb apply. In contrast, for *vertical outsourced learning*, clients perform Protect locally, while servers merge the vertically partitioned datasets in Aggregate before proceeding with horizontal training as in OL (Alg. 2).

Appendix G
Collaborative Labeling

CAPC [84] augments the DP approach Private Aggregation of Teacher Ensembles (PATE) by Papernot et al. [43] with cryptography. In PATE, a *student* (St) queries a set of *teachers* (T_i) to collaboratively label its dataset using their local models. The goal of collaborative labeling is not to train a DP model but to label a dataset for local training. The protocol output is the predicted label set via majority voting. The labeling needs to be executed securely, i.e., with encrypted data, since the St data need to remain private, and the teachers' models need to be protected from inference attacks (App. P), i.e., applying PerturbLabel. Specifically, in CAPC, St encrypts its data and performs secure inference with each T_i via a 2-party mixed-protocol [121], i.e., combining FHE [51] and MPC [122]. From the inference, each T_i receives a homomorphically encrypted label, masks it (i.e., adds random value), and sends to St. Thus, the label is secret shared with St, i.e., St learns the masked value after decryption and T_i knows the mask. Now, St and T_i compute the *one-hot encoding* of the label via GC [123], secret sharing the result. Afterwards, each T_i sends its share of the (one-hot encoded) label to a third party, called *privacy guardian* (PG), which applies PerturbLabel, and runs GC [123] with St to compute the label with the most votes.

Appendix H
Quantization techniques

Here, we describe the advantages and disadvantages of employing quantization and working with discrete values. Quantization approximates d-dimensional vectors from $\mathbb{R}^d$ to $\mathbb{Z}^d$. This can reduce the size of messages exchanged among parties, decreasing the communication costs. On the other side, quantization introduces an approximation error that leads to a trade-off between communication and accuracy, since larger number of bits mean more accuracy but also more communication. Next, we provide a brief description of the quantization methods applied to secure aggregation, by two of the analyzed works. Agarwal et al. [10] apply stochastic k-bit quantization composed with binomial mechanism. The data are preprocessed with random rotation to reduce the leading quantization error term according to Suresh et al. [124]. They show that this technique not only reduces the approximation error, but also improves the privacy guarantee. In [10, Corollorary 3] they state that their protocol achieves the same error and privacy of the full precision Gaussian mechanism with a total communication costs of $O(nd \log\log(nd/\delta\epsilon))$ when $d = O(n\epsilon^2)$ instead of $O(nd \log nd)$ for d variable and n clients. Kairouz et al. [7] observe that the communication cost depends on the dimensionality d of inputs and $\log m$, which is the number of bits per coordinate. Since they can not control d, the only factor that can be reduced is $\log m$. Kairouz et al. [7] scale and clip the input vectors, through rotations/reflections,

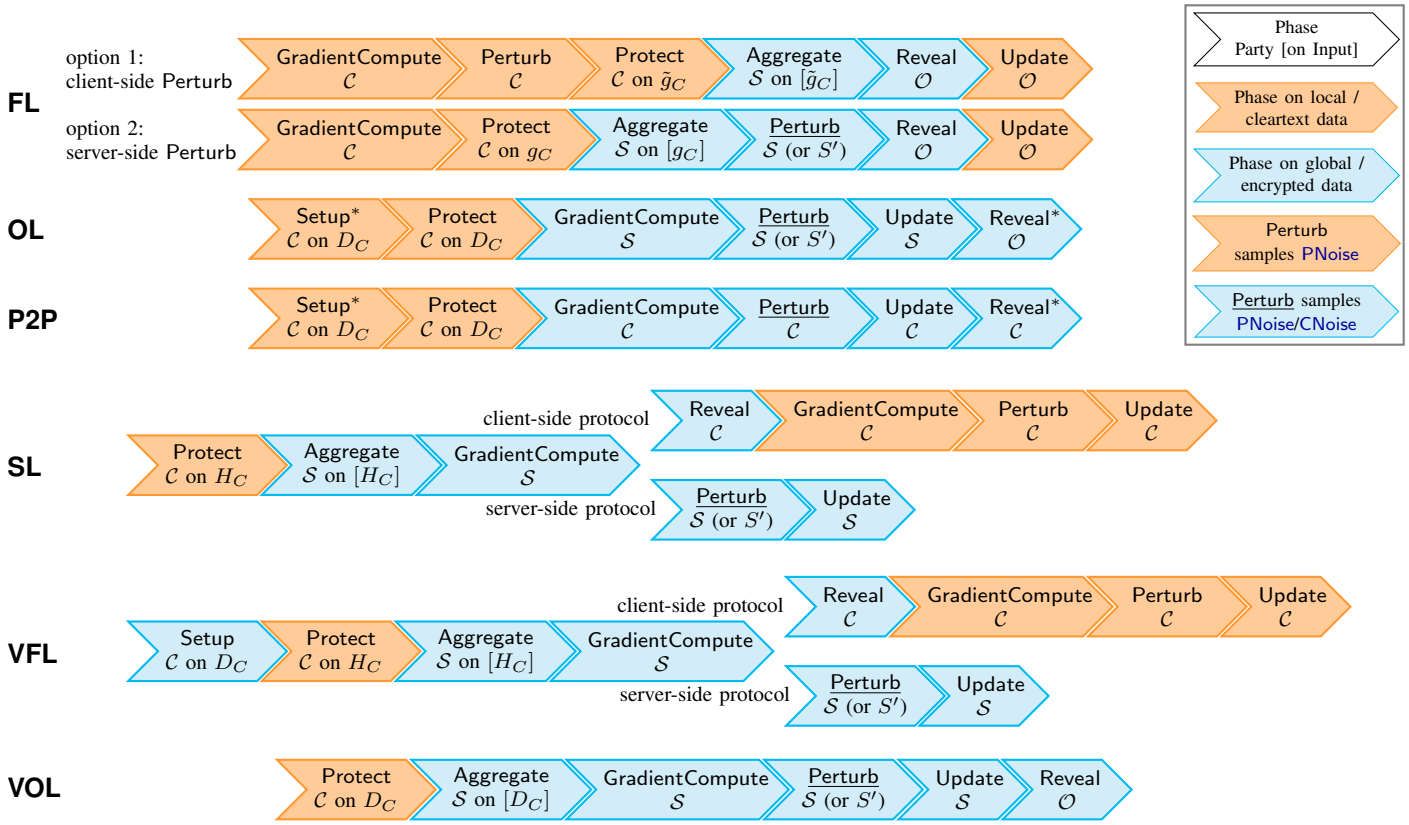


Fig. 2: Overview of the CPCL phases across various learning paradigms, showing phase order, party roles, inputs to phases where relevant, and if computations are local or global. The legend in the top-right corner clarifies box styles and color coding. Perturb means that either PNoise can be sampled locally, or CNoise via MPC or a semi-trusted server. The Setup phase is omitted unless it involves specific actions, i.e., entity alignment in VFL and local data pre-processing in OL and P2P. Optional phases are marked with superscript $*$. Inputs to Aggregate and Protect phases vary by paradigm: gradients g for FL, intermediate activation outputs H for CFL and SL, and dataset D_C from clients in VOL and OL. The Perturb phase applies to gradients in all paradigms, as we focus on PerturbGradient. Two execution paths represent possible design options for FL, i.e., client- vs noise-side Perturb (see Alg. 2), but different protocol views for VFL and SL, i.e., client- vs server-side execution.

flatten the vectors to reduce the distortions due to modular wrap around, i.e. this is caused by modulo operations in secure aggregation [46]. Each element of the input vector is randomly and independently rounded to one of the two the nearest integers. The scaling and flattening are undone on the server side during the Aggregate. They combine quantization with discrete Gaussian noise aggregation, and show that 16 bits per coordinate are sufficient to nearly match the utility of the Gaussian baseline with fixed precision of 32 bits.

Appendix I
Handling Dropouts

Dropouts are parties that prematurely leave the protocol during its execution. In single-server architectures, handling dropouts is crucial, as clients are actively involved in the protocol. With multiple servers, clients are either computing parties or they only outsource their data to computing parties (e.g., secret sharing). Dropouts cause the protocol to stop, and in distributed settings this can frequently happen, e.g., mobile devices in secure aggregation can easily drop out [46]. We can distinguish dropout resistant protocols with or without additional clients interaction. Stevens et al. [49] and Bindschaedler et al. [70] guarantee dropout resistance without additional clients interaction since both protocols have a distinct phase for secret key reconstruction based on threshold secret sharing which can terminate successfully without the local updates of the dropped clients, i.e., if less than threshold clients drop out.

Secure aggregation protocols based on Bonawitz et al. [46], e.g., Kairouz et al. [7], require additional communication rounds since the clients can dropout after the Setup phase, i.e., sharing of the random masks. To ensure confidentiality, each client adds the masks of all the clients including the dropouts, which do not cancel out in the Aggregate phase. To ensure dropouts resistance, each client secret shares its mask with the other clients, so the server can ask the clients to reconstruct the masks of the dropouts which is removed after Aggregate. This causes a privacy issue for the dropouts, since the server can maliciously mark a client as dropped out and later retrieve the local update of that client removing the reconstructed mask. Bonawitz et al. [46] implement a double masking to protect the

privacy of dropouts. Each client generates two masks which are both secret shared among the clients, i.e., one for dropout resistance and one to protect privacy. The server can asks for the reconstruction of only one mask per client. For a complete description of the protocol we refer to [46].

Recent advancements [125], [126] leverage a threshold version of the Joye-Libert (TJL) AHE scheme [127] to avoid the double masking. In KarakoC_c et al. [126] a key dealer distributes a secret key for each client and the decryption key to the server. The decryption key works only if all the clients send their encrypted update. To handle dropout, during setup, the clients secret share the encryption of zero via a (t, m)-SS scheme with other clients. If a client drops out, the other clients can collaboratively encrypt a zero on behalf of the dropped client and the server can decrypt the sum of the updates. Taiello et al. [125] extend KarakoC_c et al. [126] with a secure aggregation protocol which relies only on online clients. Each client generates a per-round JL key and protects it with a TJL key which each client secret shares among selected clients. The server reconstructs the per-round aggregation key with the collaboration of at least t online clients. The model parameters are computed using the aggregation key which is the sum of the per-round keys of online clients.

Dropouts are also an issue for partial noise aggregation, since the remaining clients have to ensure that the aggregated noise achieves CDP. As discussed in Section V-B, to also tolerate up to s dropouts, the denominator of Eq. (3) can be set to $n-(t+s)$. For examples, Dwork et al. [128] guarantees Byzantine robustness, with $2/3$ of remaining clients.

APPENDIX J
NOISE DISTRIBUTIONS

Next, we provide details on how to convert privacy parameters to distribution parameters for different noise mechanisms.

Distributed Laplace. The distributed Laplace mechanism is realized combining gamma-distributed random variables in the following way: $\mathsf{Lap}(0,\lambda) = \sum_{k=1}^{N} \mathsf{Gamma}_k - \mathsf{Gamma}'_k$, where the scale parameter $\lambda = \Delta_1/\epsilon$, and each gamma-distributed value $\mathsf{Gamma}_k, \mathsf{Gamma}'_k$ is sampled from:

$$\mathsf{Gamma}(x; 1/N, \lambda) = \frac{1}{\Gamma(1/N)\lambda^{1/N}} x^{1/N-1} \exp\left(-\frac{x}{\lambda}\right),$$

where $\Gamma(1/N) = \int_0^\infty x^{1/N-1}\exp(-x)\,dx$.

Gaussian. For the Gaussian distribution, the noise variance σ_{DP}, for $\epsilon \leq 1$, is computes as: $\sigma_{DP} \leftarrow \sqrt{2\ln(1.25/\delta)}\Delta_2/\epsilon$ For partial noise aggregation, σ_i is computed according to Eq. (3) considering σ_{DP} as target variance.

Binomial. For the multidimensional binomial distribution, i.e., data with d dimensions, the number of coin flip ($p = 0.5$) to achieve DP guarantees close to the Gaussian, can be reduced to $m \geq 8\log(2/\delta)/\epsilon^2$. The equivalence with the discrete Gaussian is for $\sigma = s\sqrt{Np(1-p)}$ Here, $s = 1/j, j \in \mathbb{N}$ is a quantization scale, set to $s \leq \sigma/(c\sqrt{d})$ [10].

Discrete Gaussian. The value of σ for the discrete Gaussian has to optimize the following [7]:

$$\epsilon = \min\left(\sqrt{\frac{\Delta_2^2}{n\sigma^2} + \frac{1}{2}\tau d}, \sqrt{\frac{\Delta_2^2}{n\sigma^2} + 2\frac{\Delta_1}{\sqrt{n}\sigma}\tau + \tau^2 d}, \frac{\Delta_2}{\sqrt{n}\sigma} + \tau\sqrt{d}\right),$$

where d is the data dimensionality, and $\tau = 10 \cdot \sum_{k=1}^{n-1} \exp(-2\pi^2\sigma^2\frac{k}{k+1})$ The σ has to be scaled according to the quantization scale [7], as discussed in Sec. V-C.

Skellam. A Skellam random variable can be sampled as the difference of two Poisson distributed random variable sampled from the Poisson distribution:

$$\mathsf{Poisson}\left(x; \frac{\mu}{2}\right) = \left(\frac{\mu}{2}\right)^x \frac{\exp(-x)}{x!}.$$

The variance μ for the Skellam distribution, to achieve (α, ϵ)-RDP, optimizes the following [62]:

$$\epsilon(\alpha) \leq \frac{\alpha\Delta_2^2}{2\mu} + \min\left(\frac{(2\alpha-1)\Delta_2^2}{4s^2\mu^2} + \frac{3\Delta_1}{2s^3\mu^2}, \frac{3\Delta_1}{2s\mu}\right).$$

Here, s is the scaling factor, and Δ_1 and Δ_2 are the l_1 and l_2 sensitivities. The variance μ needs to be rescaled according to the quantization scale [62], as discussed in Sec. V-C.

Poisson-Binomial. All mechanisms in Tab. II add noise to the gradients except for the Poisson-Binomial mechanism for PNoise [64] which encodes local gradients into a parameter of the binomial distribution.

$$\mathsf{PoiBin}(x; \mathbf{g}, b, p) = \mathsf{Bin}(x; b, \mathbf{g}\theta/K + 1/2).$$

Here, $\theta \in [0, 1/4]$, and b is the output bit-width. The mechanism first applies a rescaling of the input gradient $\mathbf{g}$, and then uses the result as the prob of binomial. Since this mechanism is optimized for l_∞ geometry, each client computes the Kashin representation of g_i [129], i.e., transforms the geometry of the data from the l_2 to l_∞. The server reverts the Kashin representation to l_2 geometry after the aggregation.

A. Noise Sampling Algorithms

We formalize sampling algorithms in Alg. 3. Specifically, we provide the pseudocode to sample in cleartext BoxMuller, LaplaceITS, Skellam, DiscLaplace and DiscGauss distributions with related subroutines, i.e., Poisson, Geom and Bern. Before proceeding, we detail basic sampling concepts.

Basic Sampling Algorithms. *Inverse transform sampling* (ITS) allows sampling random variables $\Psi \sim f(x)$ from a probability distributions $f(x)$ using a uniformly distributed random variable $u \sim U(0,1)$ in $(0,1)$ [88]. ITS sample Ψ via the inverse *cumulative density function* of $f(x)$ $\Psi = CDF^{-1}(u)$. For example, to sample from a Laplace distribution LaplaceITS (Alg. 3) computes $-\frac{1}{\lambda}\,\mathsf{sign}(u-0.5)\log u$ with sign function $\mathsf{sign}(x)$. However, the Gaussian's CDF^{-1} has no closed form and needs to be approximated, e.g., via Taylor polynomials [130]. Alternatively, the BoxMuller *transform* [89] (Alg. 3) receives two $u_1, u_2 \sim U(0,1)$ and outputs two samples from the standard Gaussian $\mathcal{N}(0,1)$.

APPENDIX K
NOISE SAMPLING SCENARIOS

In Fig. 3, we depict the noise sampling scenarios for CNoise and PNoise from Alg. 2. The four noise scenarios are

Input: Tuple of noise parameters τ. For each mechanisms, the parameters are: $u, u_1, u_2 \sim U(0,1)$, scale λ (Tab. II), mean μ and scaling factor s.

Func Sample (τ)**:**
  NoiseType, Mechanism, params $\leftarrow$ Parse (τ) `// From` τ `parse the tuple of noise parameters, e.g.,` $\tau = (\text{PNoise}, \text{DiscGauss}, (s, \sigma))$
  return Mechanism(params) **if** NoiseType $==$ PNoise **else** Mechanism(params) `// Call the sampling function, e.g.,` DiscGauss $(s^2\sigma^2)$

Mech BoxMuller (u_1, u_2, σ)**:**
  $\mathcal{N}_1 \leftarrow \sqrt{-2\log(u_1)}\cos(2\pi u_2)$
  $\mathcal{N}_2 \leftarrow \sqrt{-2\log(u_1)}\sin(2\pi u_2)$
  return $\sigma\mathcal{N}_1, \sigma\mathcal{N}_2$

Mech LaplaceITS (λ, u)**:**
  return $-\frac{1}{\lambda}\,\text{sign}(u - 0.5)\log u$

Mech DiscGauss $(s^2\sigma^2)$**:**
  $b \leftarrow 2s^2\sigma^2$
  do
    $l \leftarrow$ DiscLaplace $(s\sigma)$
    $a \leftarrow (|l| - s\sigma)^2$
  while Bern $(a, b) == 1$
  return l

Mech DiscLaplace (λ)**:**
  $g_1, g_2 \sim$ Geom $(1 - \exp(-\frac{1}{\lambda}))$
  return $g_1 - g_2$

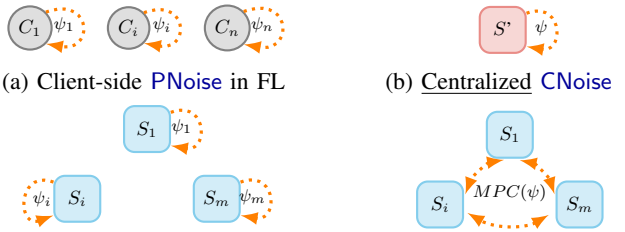
Mech Skellam $(s^2\mu)$**:**
  $p_1, p_2 \sim$ Poisson $(s^2\mu/2)$
  return $p_1 - p_2$

Func Poisson (μ)**:**
  $prod \leftarrow 1, p \leftarrow -1$
  while $prod > \exp(-1/\mu)$ **do**
    $p \leftarrow p + 1$
    $u \sim U(0,1)$
    $prod \leftarrow prod \cdot u$
  return p

Func Bern (a, b)**:**
  $u \sim U(0,1)$
  return $u < \exp(-\frac{a}{b})$

Func Geom (p)**:**
  $u \sim U(0,1)$
  return $\lfloor \frac{\log(1-u)}{\log(1-p)} \rfloor$

Algorithm 3: Noise sampling algorithms (Sample, Alg. 2). DP to distribution parameters conversion in Tab. II and App. J.



(a) Client-side PNoise in FL (b) Centralized CNoise

(c) Server-side PNoise (d) Distributed CNoise

Fig. 3: Noise sampling scenarios from Alg. 2. Dotted arrows (⋯►) indicate noise sampling.

instantiations of the three noise sampling techniques analyzed in Sec. V-A, V-B, V-C.

Appendix L
Details on Evaluation Setup

For all the evaluations (Sec. V-C) we used AWS t2.xlarge instances with 4 vCPUs (based on Intel Xeon) and 16GB RAM. In the LAN setting, with the two machines in Frankfurt (Germany), we have about 1 Gbps bandwidth and 27.6 ms latency. In the WAN setting, with one machine in Frankfurt (Germany) and the other in California (USA), we have about 74 Mbps bandwidth and 287.7 ms latency.

MPC overhead (Sec. V-C). For the evaluation of distributed CNoise sampling with MP-SPDZ [95] in Sec. V-C, we used 32-bit fixed-point protocols for continuous distributions, i.e., LaplaceITS and Gaussian via BoxMuller, and 16 bit integer protocols for discrete ones, i.e., DiscGauss and Skellam, since according to the evaluation of Agarwal et al. [62] 16 bits are enough to match the accuracy of continuous Gaussian. We used standard MP-SPDZ parameters, i.e., 40 bit statistical security and 128 bit security parameter.

Accuracy Evaluation (Tab. V, Sec. VI). We used the 3-layers neural network from [37], [73] with $768 - 100 - 10$ neurons for a total of 79510 parameters. We used ReLU activation and cross-entropy loss with DP-SGD optimizer with no momentum. We evaluate the accuracy on two common benchmarking datasets for image classification: MNIST [131] and Fashion MNIST [132], which contain grayscale images of handwritten digits and clothing, respectively. Both datasets have 60k training data images of size 28×28 with 10 classes. We rely on Poisson subsampling and the privacy loss accounting tool from the PFL library [97]. For OL, we select $B = 500$, $K = 4.0$ and $\eta = 0.1$ according to Abadi et al. [37]. For FL, we subsample $n = 100$ clients from 1000 clients for each round. From our hyperparameters search in App. O-A, we found that the best parameters for FL are $K \in \{0.3, 0.5\}$, $\eta = 0.1$ and $B = 60$.

TABLE IX: Cost of online and offline phase for basic MPC protocols as running time (in ms/s for LAN/WAN, resp.) and communication per party (same unit for LAN/WAN).

Online	$\log_2(\cdot)$	$2^{(\cdot)}$	$\sin(\cdot)$	$\cos(\cdot)$	$\sqrt{\cdot}$
LAN/ms	278.6 ± 5.7	278.3 ± 5.6	154.6 ± 6.5	154.5 ± 5.4	351.7 ± 11.3
WAN/s	14.14 ± 0.03	5.89 ± 0.01	8.13 ± 0.01	8.11 ± 0.02	19.56 ± 0.04
KB	20.9	5.6	6.3	6.3	21.2
Offline	$\log_2(\cdot)$	$2^{(\cdot)}$	$\sin(\cdot)$	$\cos(\cdot)$	$\sqrt{\cdot}$
LAN/s	2.80 ± 0.01	3.00 ± 0.03	1.49 ± 0.01	1.49 ± 0.01	3.33 ± 0.02
WAN/s	16.50 ± 0.76	17.43 ± 0.65	10.90 ± 0.48	10.59 ± 0.26	18.93 ± 0.04
MB	54.54	55.86	27.43	27.43	64.50

Appendix M
Evaluation of Basic MPC Protocols

Table IX reports our evaluation for basic MPC protocols with the MP-SPDZ framework [95] used in Alg. 3 (App. J-A) with 32-bit fixed-point protocols. We evaluated in the same setup of Sec. V-C (detailed in App. L), in both LAN and WAN settings, with 2-party maliciously secure Mascot [96]. The offline phase requires $10^3\times$ more communication than the online one, as well as $10^3\times$ the running time for LAN. The square root is the most expensive function to compute since it requires the highest amount of communication for online and offline phases and has the longest runtime. The trigonometric functions, i.e., sin and cos, have the lowest runtime in LAN, which is half of the square root one. The exponentiation base two, i.e., 2^x, requires less communication than the other functions for the online phase, which motivates why for LAN it has the same runtime of $\log_2$, whereas for the WAN setting it has the lowest online runtime, i.e., $1/3$ of $\log_2$ one. We evaluate $\log_2$ and $2^{(\cdot)}$ as MP-SPDZ uses them as building blocks to compute $\log_b(x) = \log_b(2) \cdot \log_2(x)$ for a base b, and $x^y = 2^{y \log_2(x)}$.

Appendix N
Evaluation Details for MPC Noise Sampling

Next, we provide more details on the evaluation of MPC noise sampling algorithms (Sec. V-C) including how we set the loop count for sampling DiscGauss and Poisson with minimal failure probability, the online and offline phases in LAN and WAN settings, and the communication overhead.

TABLE X: MPC online runtime for LAN/WAN in ms/seconds.

LAN, ms	LaplaceITS	BoxMuller	DiscGauss
Shamir	68.9 ± 9.7	186.8 ± 20.3	$2,866 \pm 128$
Malicious Shamir	73.4 ± 14.5	167.3 ± 24.4	$2,422 \pm 132$
Mascot	143.0 ± 7.29	425.4 ± 13.4	$5,882 \pm 55$
WAN, s	LaplaceITS	BoxMuller	DiscGauss
Shamir	5.76 ± 0.02	17.90 ± 0.05	253.76 ± 0.32
Malicious Shamir	5.39 ± 0.04	16.04 ± 0.07	219.59 ± 0.21
Mascot	17.19 ± 0.02	53.23 ± 0.09	703.58 ± 1.92

TABLE XI: MPC offline runtimes for LAN/WAN in ms/seconds per party

LAN, ms	LaplaceITS	BoxMuller	DiscGauss
Shamir	4.02 ± 1.60	9.94 ± 2.52	147.07 ± 8.68
Mal. Shamir	42.02 ± 7.43	52.80 ± 4.79	627.1 ± 21.88
Mascot	3335 ± 26	10142 ± 35	$176,427 \pm 210$
WAN, s	LaplaceITS	BoxMuller	DiscGauss
Shamir	0.45 ± 0.001	0.85 ± 0.05	7.88 ± 0.18
Mal. Shamir	1.97 ± 0.03	2.53 ± 0.07	12.99 ± 0.14
Mascot	17.72 ± 0.40	53.97 ± 1.56	934.53 ± 7.55

TABLE XII: MPC communication per party for online/offline phase in KB/MB, resp.

Online (KB)	LaplaceITS	BoxMuller	DiscGauss	Skellam
Shamir	11.01	25.74	411.4	93,112
Mal. Shamir	41.31	93.54	1,494	346,099
Mascot	23.20	55.44	862.72	183,515
Offline (MB)	LaplaceITS	BoxMuller	DiscGauss	Skellam
Shamir	0.08	0.26	1.49	642.08
Mal. Shamir	0.50	1.20	19.92	3,395
Mascot	60.74	184.75	3,219	32,616

A. Sampling Threshold

To ensure MPC constant runtime, we replace all **while** loops of Alg. 3 (App. J-A) with **for** loops with a fixed number of iterations. To find a loop count satisfy an empirical failure probability of at most 10^{-6}, we iteratively increased the counts, until sampling succeeded without any failure for 10^6 runs. Given such a count candidate, we repeated the sampling (with 10^6 runs) 100 times and computed the average number of failures with 95% confidence interval. For DiscGauss we fixed the number of iterations to 10, obtaining on average 0.78 ± 0.21 failures, whereas Poisson with $\mu = 10$ requires at least 29 iterations with 0.74 ± 0.20 failures on average, and we set it to 30.

B. Online & Offline Phases in LAN/WAN

Recall, MPC has a data-independent offline phase, to precompute material for the online phase, and a data-dependent online phase. Next, we report the online and offline runtimes for the noise sampling techniques implemented in MP-SPDZ from Alg. 3 (App. J-A).

Online Phase. Tab. X reports the online runtimes in LAN and WAN settings for the noise sampling techniques implemented in MP-SPDZ from Alg. 3 (App. J-A). Semi-honest and malicious Shamir have comparable runtimes, while Mascot is at least $2\times$ slower (for DiscGauss in LAN). The continuous sampling techniques are at least $10\times$ faster than the discrete ones. The LAN runtime for all protocols is $10^2\times$ smaller than the WAN due to lower communication latency in a LAN setting. We omit Skellam in the table, as its online runtime is disproportionate, i.e., about 20 minutes in a LAN.

Offline Phase. Tab. XI reports the offline runtimes per party in LAN and WAN settings for the noise sampling technique implemented in MP-SPDZ from Alg. 3 (App. J-A). The offline runtime of the continuous sampling techniques is at least $10\times$ faster than the discrete ones, as they are not based on iterative methods. The LAN runtime for all protocols is $10^2\times$ smaller than the WAN due to lower communication latency in a LAN.

We omit Skellam in the table, due to its comparatively much larger overhead. In the LAN setting, Skellam has the slowest offline phase, taking on average, 43.5 seconds for Shamir, 3.5 minutes for Malicious Shamir (Mal. Sh.) and about 9 hours for Mascot, due to the high number of iterations required in our implementation.

C. Communication

Tab. XII shows the per-party communication for the online and offline phase for the noise sampling techniques implemented in MP-SPDZ from Alg. 3 (App. J-A). The discrete sampling techniques require at least $10\times$ more communication than continuous ones. The technique that requires more communication for both online and offline phases is Skellam, i.e., at least $10\times$ more than DiscGauss and $10^3\times$ more than the continuous techniques. LaplaceITS is the most communication efficient technique.

APPENDIX O
ACCURACY EVALUATION

In this section, we report the results of the hyperparameter search for FL experiments on the MNIST dataset and the impact of different hyperparameters for OL with a CNN model on the EMNIST dataset. Finally, we evaluate how the different collusion thresholds affect the accuracy of PNoise on the EMNIST dataset.

A. FL Hyperparameter Search

Next, we report the results from our hyperparameter search for FL training on the MNIST dataset. Compared to plain training (i.e., no DP or cryptography), FL introduces three additional hyperparameters: the number of local iterations per client, the local clipping threshold, and the number of clients per iteration. First, we describe the evaluation setup and then analyze the relationship between learning rate and clipping threshold. Finally, we discuss how accuracy varies with the number of local iterations.

Evaluation Setup. We trained the 3-layers neural network from [37], [73] and used the DP-SGD optimizer described in App. L on the MNIST dataset. Following systematized works [7], [62], [63], we leverage the federated averaging algorithm (FedAvg) [133] which allows clients to perform multiple local iterations and send only the delta over their local model parameters to the server. We split the MNIST dataset among 1000 clients, resulting in 60 samples for each client. We subsample $n = 100$ clients per FL iteration and vary the number of local iterations in $\{1, 2, 5\}$. We run the training for 100 epochs, with clipping threshold $K \in \{4.0, 1.0, 0.5, 0.3, 0.1, 0.03, 0.01\}$ and

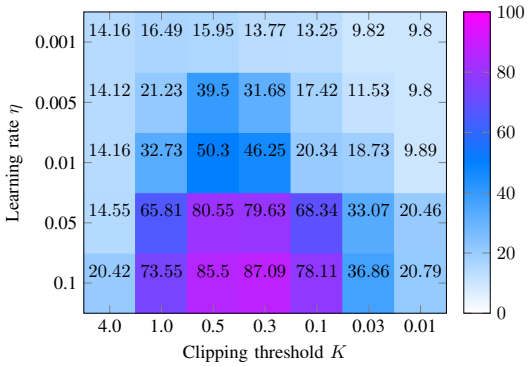


Fig. 4: FL accuracy on MNIST for different K and η fixing the number of local iterations to 5, epochs to 100, and $\epsilon = 1.0$.

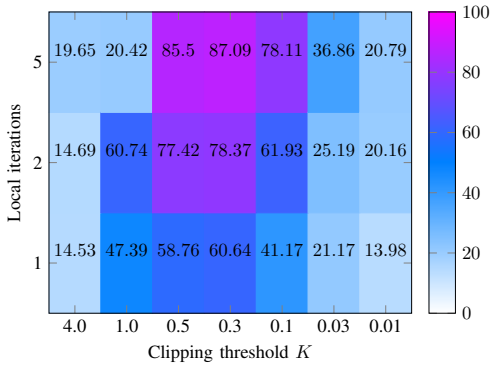


Fig. 5: FL accuracy on MNIST for different K and local iterations fixing $\eta = 0.1$, epochs to 100, and $\epsilon = 1.0$.

learning rate $\eta \in \{0.1, 0.05, 0.01, 0.005, 0.001\}$. In this analysis, we focus on $\epsilon = 1.0$ with CNoise, i.e., no additional noise variance introduced to mitigate collusion as with PNoise.

Impact of Clipping Threshold and Learning Rate. Fig. 4 shows how accuracy varies across different combinations of η and K while fixing the number of local iterations to 5. The results highlight that the learning rate significantly impacts the model accuracy, with a $\eta = 0.1$ achieving the best accuracy over all clipping thresholds. A too small $\eta = 0.001$ leads to accuracy drops of over 70 pp compared to $\eta = 0.1$, with accuracy close to random guessing across all clipping thresholds. The clipping threshold reduces the impact of noise on the gradient updates since the DP noise variance scales with K, i.e., $\sigma_{\text{DP}} = \sigma K$. Too large $K = 4.0$ or too small $K = 0.01$ lead to poor performance since either the noise variance is too high or the gradient update is too small to ensure convergence. Both show 65 pp lower accuracy than the best clipping threshold for both $K = 4.0$ and $K = 0.01$. $K \in \{0.5, 0.3\}$ and $\eta = 0.1$ achieve the best result, with less than 2 pp accuracy difference between the two K.

Impact of Local Iterations and Clipping Threshold. Fig. 5 reports the accuracy for different numbers of local iterations and clipping thresholds while fixing $\eta = 0.1$, which is the best-performing learning rate according to Fig. 4. By varying the number of local iterations, the best performing clipping thresholds remain $K \in \{0.5, 0.3\}$, since they balance noise variance and gradient updates to ensure convergence. Increasing the number of local iterations improves accuracy and reduces FL communication bottleneck due to aggregation, as analyzed in Sec. VI. More local iterations, as in FedAvg, allow for more refined local updates and reduce the impact of noise since noise is added over multiple iterations. We achieve the best accuracy with 5 local iterations, i.e., 87.09% accuracy, which is 10 pp higher than 2 local iterations and 27 pp higher for 1 local iteration. Higher accuracy with fewer local iterations requires more FL iterations, increasing communication overhead. For example, with 1 local iteration, accuracy improves to 80.82% after 1000 FL iterations, still 6 pp lower than using 5 local iterations with only 100 FL iterations. Similarly, with 2 local iterations, accuracy reaches 83.99% after 500 FL iterations, 3 pp below the best setting.

B. OL Evaluations with CNN

Next, we evaluate how the clipping threshold and the learning rate affect the accuracy of simulated OL training. First, we introduce the evaluation setup and then discuss results.

Evaluation Setup. We conduct the hyperparameter search in this section by simulating OL training with DP-SGD via PyTorch. We use the CNN model for image classification from LEAF with 6603710 parameters, as it is a standard benchmark for distributed settings [134]. We conduct our evaluation on the EMNIST dataset [135], as [7], [62], to classify handwritten characters, i.e., 62 unbalanced classes. The EMNIST dataset contains $N = 814255$ samples with different handwriting styles, which can simulate collaborative settings, i.e., about 3500 users [134]. We applied the same transformation as LEAF [134] and Agarwal et al. [62] to the EMNIST dataset, i.e., rescaling the pixel values from (0,255) to (0,1) and setting 1 for the background and 0 for black pixels. We run the training over 300 epochs with a batch size of $B = 836$ according to [37] (B=$\sqrt{N}$). We used Gaussian noise with Poisson subsampling. We vary the learning rate $\eta \in \{0.15, 0.1, 0.05, 0.01, 0.001\}$ and the clipping threshold $K \in \{10, 5, 3, 1\}$.

Impact of Clipping Threshold and Learning Rate. Fig. 6 reports the accuracy on the EMNIST dataset for different clipping thresholds and learning rates. We achieved the best accuracy with a learning rate of 0.15 and a clipping threshold of $K = 3$. The results show that by fixing η, the K has a significant impact on the model accuracy, since a too large clipping threshold, ($K = 10$), leads to a model with poor performance, i.e., accuracy of 5% since the noise standard deviation is too high, covering the gradient signal. On the other hand, when the clipping threshold is smaller ($K = 1$), the model performance starts to decrease, i.e., 2% lower compared to $K = 3$, since we clip too much of the gradient signal.

By fixing the number of training epochs, the learning rate has a significant impact since a learning rate that is too small leads to poor performance. For example, with a learning rate of 0.001, the model accuracy is 5.81%, which is 71.34% lower compared to the best learning rate of 0.15, with $K = 3$. Notably, decreasing the learning rate for $K = 10$ does not achieve the expected results. Since it achieves the best

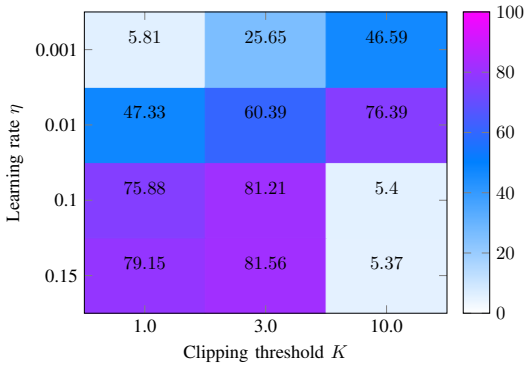


Fig. 6: Accuracy on EMNIST for different learning rates and clip thresholds for $\epsilon = 1.0$ over 300 epochs

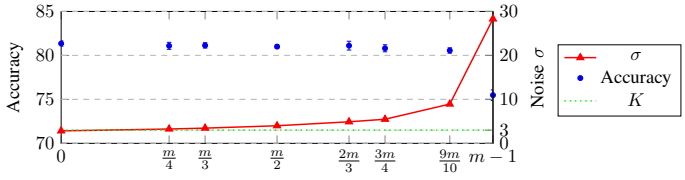


Fig. 7: Accuracy of OL with PNoise under varying collusion thresholds (t). Simulation over 300 training epochs with $\epsilon = 1$, $\eta = 0.15$, $K = 3.00$, $B = 836$, $m = 100$, with Gaussian noise (starting from $\sigma = 2.83$).

accuracy (76.39%) with $\eta = 0.01$, which is more than 70% higher than the accuracy with $\eta = 0.15$. Then for $\eta = 0.001$ the accuracy drops to 46.59%. This is probably because a smaller learning rate compensates for the large noise variance.

C. Collusion Effect on PNoise

Next, we evaluate the impact of collusion on PNoise training by varying the collusion threshold t and analyzing its effect on accuracy and noise variance. First, we describe our evaluation setup and then present our results.

Evaluation Setup. We trained the same CNN as in App. O-B on the EMNIST dataset using DP-SGD. We vary the collusion threshold $t \in \{0, m/4, m/3, m/2, 2m/3, 3m/4, m-1\}$. We set $\epsilon = 1.0$ and use the best hyperparameters from App. O-B. We simulate a PNoise setting with $m = 100$ using PyTorch [87]. Fig. 7 reports the average accuracy over 10 runs by taking the maximum for each run over the last five epochs to account for randomness in noise sampling.

Accuracy Impact. Fig. 7 shows the impact of varying t on accuracy and noise standard deviation σ according to Eq. (3). Interestingly, the accuracy remains stable until $t = 9m/10$, despite a $3\times$ increase in σ. Although σ is greater than K, meaning the noise has a higher probability of overshadowing gradient updates, the results suggest that gradient directions remain largely unaffected. This may explain why none of the systematized works implement oblivious PNoise protocols with PerturbGradient. However, when $t = m - 1$, accuracy drops by approximately 10% compared to $t = 0$.

Appendix P
Privacy Attacks in CL

Next, we discuss privacy risks of cryptography-only CL in various threat models, focusing on membership inference (MIA), gradient inversion (GIA), and DP noise tampering attacks. We overview DP and cryptographic mitigations, while addressing DP side-channels. We focus on attribute inference attacks in App. P-A.

Threat Models. OL operates in black-box settings, where adversaries access only model outputs and datasets of corrupted $\mathcal{C}$, since $\mathcal{S}$ train on encrypted data. Here, active attackers can tamper with local datasets and manipulate noise protocols. FL operates in white-box settings, where $\mathcal{C}$ and $\mathcal{S}$ access model parameters and gradients. Here, an active attacker can also alter gradients. Next, we discuss black-box attacks and then risks of revealing gradients.

MIA. Membership inference attacks (MIA) aim to infer whether a specific record (or user) was part of the training data by exploiting differences in model behavior, e.g., confidence scores [4]. Black-box attacks typically use a shadow dataset (i.e., similar in distribution to the target dataset) to train an attack model to distinguish members [136]. White-box access to gradient patterns can improve attack accuracy by up to 7% [136]. Active adversaries can perform *poisoning MIA* (PMIA) to amplify MIA's effects by corrupting target models [137]. In OL, $\mathcal{C}$ can inject mislabelled target samples, causing the model to treat correctly labeled samples as outliers. Reportedly, 8 poisoned samples in CIFAR-10 can improve MIA's true positive rate by 52 pp [56]. In FL, corrupted $\mathcal{C}$ can alter local parameters to increase the loss on a target record x. If another client has x, the tampered local SGD sharply reduces its gradient norm [136], [137], boosting attack accuracy by 10 pp over black-box [137]. Poisoning can also embed privacy-backdoors in pretrained models to facilitate MIA. During pretraining, models memorize target samples, and reinforce or forget them in fine-tuning, i.e., depending on their presence in the victim's dataset [138].

Mitigating MIA/PMIA via DP and Cryptography. DP-SGD effectively mitigates MIA and PMIA by bounding the influence of any data point, even a poisoned one, on the model [56], [57]. For example, a small amount of noise (e.g., $\epsilon > 5000$, $K = 10$) reduces MIA effectiveness to random guessing [139]. However, mitigating PMIA requires stricter privacy guarantees leading to accuracy drop, e.g., by 15 pp with 1000 poisoned samples ($\epsilon \approx 3$) [140]. DP-SGD is also effective against privacy-backdoors, but requires strong privacy guarantees ($\epsilon \leq 8$) [141]. While relying solely on DP introduces a privacy-accuracy trade-off, augmenting DP with cryptography can enhance model utility and robustness. In FL, zero-knowledge proofs (ZKP, App. E-C) can ensure gradients are within valid ranges without revealing the values [2]. Alternatively, robust secure aggregation identifies malicious updates, down-weighting them during aggregation; e.g., CAFCOR [68] identifies updates that disproportionately increase the collective variance. In OL, servers can implement

MPC variants of outlier detection, regularization techniques [140], or poisoning-resistant training algorithms [142].

Noise Tampering and Cryptographic Mitigations. Active adversaries can manipulate noise protocols, weakening DP guarantees. Non-oblivious PNoise approaches pre-adjust variance to account for up to t zero-noise additions (Sec. V-B), at utility cost (Tab. V). To preserve model utility and mitigate noise tampering oblivious protocols need to be combined with verifiable noise sampling. Verifiable sampling requires clients (or servers) to provide proofs of correct sampling of PNoise (or CNoise), e.g., via ZKPs [130]. However, cryptographic proofs introduce significant overhead, e.g. up to hours to prove one sample via BoxMuller [130]. Discrete sampling methods incur even higher costs due to their iterative nature. For distributed CNoise sampling, maliciously secure protocols ensure correct computation but incur additional overhead, e.g., up to 7× slower than semi-honest in a WAN (Tab. III). Verifiable and oblivious noise selection can be achieved via verifiable shuffling [34], or ZKPs. In client-side protocols, ZKPs ensure selection of one sample from a proposed list, while in server-aided protocols, ZKPs verify that exactly one sample per client has been selected [70].

DP Side-Channels and Cryptographic Mitigations. DP implementations can introduce side-channels undermining DP guarantees. Sampling continuous noise on finite-precision machines can lead to *floating-point attacks* [143], [144]. The intuition is that finite-precision machines can not represent all possible values, revealing holes in distributions [145]. As a result, some DP outputs occur only for certain inputs. Similarly, *timing attacks* infer the noise magnitude from the sampling runtime [143]. Fixed-point and quantized integer representations typically used in cryptographic protocols for DP inherently mitigate DP side-channels [144]. Thus, CPCL works are secure against those attacks. Furthermore, constant-time MPC protocols for CNoise [38], [71], [72] mitigates timing attacks, by avoiding data-dependent leaks (Sec. V-C). Caching offline-sampled noise is a possible mitigation for PNoise and centralized CNoise. Moreover, CPCL samples multiple noise values per epoch, attackers only observe cumulative times, reducing attacks' effectiveness [143].

GIA in FL and Cryptographic Mitigations. White-box access enables gradient inversion attacks, where adversaries reconstruct training samples by optimizing a loss function to match observed gradients [58]. Furthermore, an active attacker who corrupts the server can recover training samples by injecting convolutional layers that enable separation of client gradients after aggregation [146]. While DP mitigates MIA, GIAs can still be feasible even with LDP [58]: a corrupted server can reconstruct training samples with a model trained on auxiliary data ($\epsilon \approx 10$ and $n \leq 4$). Notably, large $\epsilon \approx 10^3$ is ineffective against a malicious server, while small $\epsilon \approx 0.1$ reduce the success to near-random guess [146]. To mitigate GIA, FL can leverage cryptographic techniques to guarantee gradient and model secrecy against $\mathcal{S}$. HE with a single server [59] or SS with multiple servers ensure that $\mathcal{S}$ operate on encrypted data, and only $\mathcal{C}$ learn gradient updates. This guarantee can be extended to active adversaries by using ZKPs to prove correct aggregation [34].

A. Attribute Inference Attacks

In attribute inference attacks (AIA), an adversary with partial knowledge of a record attempts to infer unknown attributes using a model trained on that record [147]. Unlike MIA, which determine whether a record was in the training set, AIAs leverage statistical correlations in the data to infer sensitive attributes [147]. AIAs pose an additional risk as ML models can learn unintended information beyond their original task. For example, a model trained to predict age from profile photos may inadvertently learn to infer race [148], [149]. These attacks are particularly effective in white-box settings, where the adversary has not only access to model parameters, but also to intermediate representations of a target sample, e.g., embeddings [149]. For example, in scenarios where the model is split into embeddings and a classifier, the clients can locally compute embeddings and send the result to a server for classification. However, the adversary can exploit the embeddings to launch an AIA, as the embeddings may over-learn sensitive features during training [148].

Discussion on DP mitigations for AIA. Theoretically, DP protects against AIA, as the DP adversary is stronger and assumes access to more information than typical AIA adversaries, who only have partial knowledge of a record [108]. Specifically, a DP adversary seeks to distinguish between models trained on neighboring datasets (i.e., differing in at most a record/user), while an AIA adversary uses partial knowledge of a record to infer unknown attributes.

Empirical evaluations show that black-box AIAs rarely learn more than an adversary could infer from prior knowledge alone. However, white-box AIAs, where an attacker has access to model parameters, can identify records with a sensitive attribute more effectively [147]. DP does not directly mitigate this risk because its guarantees focus on record- or user-level indistinguishability rather than protecting against statistical leakage from the distribution itself. In fact, DP aims to maintain population-level information while protecting individual records or users. Furthermore, while DP noise is injected into gradients to reduce the impact of a single record, the noise can also amplify correlations between neurons and sensitive attributes in some cases [147]. Thus, while strong privacy guarantees ($\epsilon < 1$) may mitigate AIAs to some extent [150], DP does not inherently prevent attackers from leveraging the statistical properties of the training distribution, since learning those properties is the goal of ML model training.

Cryptographic mitigations for white-box AIA. The use of cryptography in CPCL inherently mitigates white-box AIA based on intermediate model representations. The model trained in OL can be kept encrypted during inference, preventing the adversary from accessing intermediate representations. In FL, clients have access to local models but do not know the embeddings of other clients, preventing white-box AIA attacks [149]. In settings where the model is split into embeddings

and a classifier, the clients can encrypt the embeddings before sending them to the server for classification. This prevents the adversary from exploiting the embeddings to launch an AIA, as the embeddings are encrypted and the server works over encrypted data and model.

APPENDIX Q
SUMMARY OF RESEARCH DIRECTIONS AND OBSERVATIONS

Tab. XIII summarizes the mapping between the key observations identified in our systematization and the corresponding research directions proposed in Sec. VIII.

TABLE XIII: Mapping of Key Observations to Future Research Directions.

Research Direction	**Driving Observation(s)**
(D1) Enhance privacy and performance via pre-processing.	**(O3)** Client local pre-processing in OL reduces cryptographic overhead while maintaining high accuracy.
(D2) Provide cryptographic building blocks for DP in OL.	**(O1)** OL inherently guarantees gradient and model secrecy against both clients and servers, preventing GIA. **(O2)** There are no secure per-example clipping optimizations via cryptographic protocols. **(O9)** Efficient and accurate CPCL solutions require careful design considerations for learning paradigms, cryptographic protocols, and noise sampling techniques
(D3) Develop crypto-friendly discrete CNoise sampling.	**(O6)** The choice of noise distribution significantly affects utility and performance **(O8)** Sampling algorithms for discrete distributions have non-constant runtimes.
(D4) Embed DP outside Perturb.	**(O7)** DP can be embedded in phases besides Perturb.
(D5) Propose strong user-level DP algorithms for multi-user datasets.	**(O4)** No established best practice exists for user-level DP in multi-user datasets.