Privacy-Preserving Data Deduplication for Enhancing Federated Learning of Language Models

Aydin Abadi⋆[1], Vishnu Asutosh Dasu⋆[2], and Sumanta Sarkar⋆[3]

[1] Newcastle University
aydin.abadi@ncl.ac.uk
[2] Pennsylvania State University
vdasu@psu.edu
[3] University of Warwick
sumanta.sarkar@warwick.ac.uk

Abstract. Deduplication is a vital preprocessing step that enhances machine learning model performance and saves training time and energy. However, enhancing federated learning through deduplication poses challenges, especially regarding scalability and potential privacy violations if deduplication involves sharing all clients' data. In this paper, we address the problem of deduplication in a federated setup by introducing a pioneering protocol, Efficient Privacy-Preserving Multi-Party Deduplication (EP-MPD). It efficiently removes duplicates from multiple clients' datasets without compromising data privacy. EP-MPD is constructed in a modular fashion, utilizing two novel variants of the Private Set Intersection protocol. Our extensive experiments demonstrate the significant benefits of deduplication in federated learning of large language models. For instance, we observe up to 19.61% improvement in perplexity and up to 27.95% reduction in running time. EP-MPD effectively balances privacy and performance in federated learning, making it a valuable solution for large-scale applications.

1 Introduction

The application of machine learning (ML) has experienced rapid growth, becoming widely embraced by businesses due to its various benefits. As ML relies heavily on data, there is an inherent risk of privacy breaches associated with this process. Federated learning (FL) is one of the main steps towards privacy-preserving ML that allows training across multiple decentralized devices without exchanging data. In FL, devices compute local models based on their data and then share the local model updates with a central server. This server aggregates the updates to derive a global model that encapsulates the features of all the local data held by the individual devices [38].

The *quality* of the training data significantly influences the *accuracy* of an ML model. Data generated from real-world applications often lacks organization, leading to issues such as missing values, typos, format mismatches, outliers, or duplicated entries within the raw dataset. To ensure meaningful learning, the collected data must undergo a thorough data cleaning process [23]. Duplicated sequences are prevalent in text datasets. They can adversely affect the training process of Language Models (LMs). Lee et al. [31] investigated the Colossal Clean Crawled Corpus (C4) [44] dataset. They discovered a 61-word sequence within C4 that was repeated 61,036 times verbatim in the training dataset and 61 times in the validation set. As a result, the trained model generalized poorly on the dataset, and over 1% of the unprompted model outputs were memorized and copied verbatim from the training dataset. Upon deduplicating the dataset, they reduced memorization by up to 10× and, in certain cases, even improved perplexity by up to 10%.

Additionally, memorization negatively affects the privacy and fairness of LMs [13]. Memorization makes language models vulnerable to membership inference attacks [48,37], data extraction attacks [39,14], and can lead to copyright violations as LMs can regurgitate training data sequences from copyrighted sources [28]. Carlini et al. [13] conclude that bigger LMs memorize more and more duplicates increase memorization. Kandpal et al. [27] show that the success of most privacy attacks on LMs is largely due to the duplication in

⋆ Equal contribution. Listing order is alphabetical.

the training datasets. Furthermore, in the FL setting, malicious clients can exploit the memorization of LMs to extract sensitive information from honest clients' datasets [46]. Therefore, we must adopt data-cleaning practices to improve model utility and reduce the risks of privacy attacks.

While deduplication can improve model performance and reduce memorization, it also enhances training efficiency in various aspects [31]. Removing duplicates reduces GPU time and minimizes training costs in terms of time, money, and carbon footprint. This streamlined process optimizes resource utilization and contributes to more sustainable ML practices [8,47,42]. Sustainable development is a global priority, and aligning ML practices with this theme is crucial.

Based on the discussions thus far, it is evident that the deduplication of training data offers numerous benefits. Several real-world applications involve training with FL such as healthcare [4], smart city [25], and edge computing [51]. Some of these applications pose a risk of duplicates in the local training data across multiple devices. For example, Google Keyboard suggestions from a user's text query rely on FL [22]. Local models are trained in situ on Android phones with user data. In this setting, many text queries typed by the users across multiple phones are the same. Therefore, for a FL process to be efficient and effective, participating devices must perform deduplication. When a data owner solely intends to remove duplicates from their dataset, the privacy risk is minimal since the data is entirely under the owner's control. In the context of FL, the scenario shifts significantly when deduplication is introduced. Multiple devices participating in FL may possess overlapping data, even after deduplicating their datasets. If they aim to deduplicate the combined dataset, one approach could involve sharing their raw data with each other and checking for intersections. However, this approach compromises privacy.

Hence, there is a necessity for privacy-preserving deduplication in FL, where participating devices would collaboratively deduplicate the combined datasets in a privacy-preserving fashion. This paper demonstrates how deduplication can be executed in FL without compromising the privacy of the data belonging to the individual devices.

1.1 Overview of Privacy-Preserving Deduplication in FL

Consider nodes $D_1, \ldots, D_n$ are involved in FL, where each D_i has a dataset S_i. Effectively, the training in FL is conducted on the union of these datasets, i.e., $\bigcup_{i=1}^{n} S_i$. If intersections exist among the datasets, FL will incorporate duplicates, leading to the drawbacks discussed earlier.

Consider nodes D_1 and D_2, each with datasets S_1 and S_2, respectively implying that FL training occurs on $S_1 \cup S_2$. If the intersection $S_1 \cap S_2$ is nonempty, training individually on S_1 and S_2 would mean training twice on the duplicate $S_1 \cap S_2$. Thus, one of D_1 and D_2 should remove the duplicate $S_1 \cap S_2$. This should be done without harming each other's data privacy. Our solution applies private set intersection (PSI) which securely finds the intersection of S_1 and S_2 without revealing any other elements. Once D_1 and D_2 learn $S_1 \cap S_2$ through PSI, D_1 will train on $S_1' = S_1 \setminus S_1 \cap S_2$ and D_2 will train on S_2. According to set theory, it holds that $S_1' \cup S_2 = S_1 \cup S_2$. Hence, the resulting FL model remains as intended, while the training process is devoid of duplicates, avoiding the associated drawbacks. The scenario with two nodes seems straightforward. However, when more than two nodes participate, it becomes complicated. We now consider n nodes $D_1, \ldots, D_n$, where each node D_i has a set S_i, $\forall i, 1 \leq i \leq n$, and $n > 2$.

Following the approach used in the two-node case, one might be tempted to (i) apply multi-party PSI to n nodes, (ii) find their intersection I_n, and (iii) let node D_1 train on $S_1' = S_1 \setminus I_n$, and rest of the nodes train on their own dataset S_i. This method removes the duplicates that exist across *all* the nodes. However, this does not detect and remove the duplicates that exist among a subset of nodes. For instance, if there is a subset of k nodes ($k < n$) with a large intersection I_k, then I_k will remain in the full training dataset $S_1' \bigcup (\bigcup_{i=2}^{n} S_i)$ as duplicates. A generic multi-party PSI does not help in this case. Therefore, we need to consider each pair of nodes and remove the duplicates accordingly.

1.2 Our Contributions

In this paper, our end goal is to develop a scheme such that allows multiple clients to benefit from deduplication while training on their data in a federated learning setup. The first requirement is an efficient deduplication of sets belonging to multiple clients. To address this, we introduce the notion of *Privacy-Preserving Multi-Party Deduplication* (P-MPD) in Section 4. This essentially describes a functionality that takes input datasets $S_1, \ldots, S_m$ (potentially containing duplicates) from m clients and outputs the datasets $S'_1, \ldots, S'_m$ such that $\bigcup_{i=1}^{m} S'_i = \bigcup_{i=1}^{m} S_i$, where $S'_i \cap S'_j = \emptyset, i \neq j$. We realize that an efficient construction of P-MPD requires a substantially improved PSI protocol that supports scalability. This leads us to introduce a new notion for PSI which we call *Group PSI* (G-PSI) in Section 3. The functionality of G-PSI takes sets from a group of clients and returns each client the intersection of their sets with all the other clients' sets. We provide constructions of *Efficient Group PSI* (EG-PSI) that efficiently realizes G-PSI, namely EG-PSI$^{(\mathrm{I})}$ and EG-PSI$^{(\mathrm{II})}$ in Figures 1 and 2, respectively. EG-PSI$^{(\mathrm{I})}$ is based on symmetric key primitives such as pseudorandom permutation, while EG-PSI$^{(\mathrm{II})}$ is developed using an oblivious pseudorandom function, which is a well-known public key primitive. With the building block EG-PSI, we build *Efficient Privacy-Preserving Multi-Party Deduplication* (EP-MPD) that realizes P-MPD as shown in Figure 4. We prove the security of EP-MPD, EG-PSI$^{(\mathrm{I})}$, and EG-PSI$^{(\mathrm{II})}$ within the simulation-based paradigm. Our construction of EP-MPD allows for efficient duplicate removal without compromising clients' data privacy, resulting in an improved model after running federated learning on the deduplicated datasets, as outlined in Figure 5.

We perform an extensive experimental evaluation to benchmark EP-MPD and the effect of the resulting deduplication on FL. The overall running time for EP-MPD$^{(\mathrm{I})}$, which employs EG-PSI$^{(\mathrm{I})}$ (symmetric key primitives based), is much less than EP-MPD$^{(\mathrm{II})}$, which utilizes EG-PSI$^{(\mathrm{II})}$ (public key primitives based). Overall, clients enjoy relatively less computation time in EP-MPD$^{(\mathrm{II})}$. Our protocols can scale to large datasets and client counts. For example, when 50 clients have 2^{19} data points in their datasets comprising of 30% duplicates, then EP-MPD$^{(\mathrm{I})}$ takes 1160 seconds and EP-MPD$^{(\mathrm{II})}$ takes 7653 seconds; client running time is 641 and 111 seconds in EP-MPD$^{(\mathrm{I})}$ and EP-MPD$^{(\mathrm{II})}$ respectively. We experiment with 7 datasets, 2 language models, and 10 clients in FL. We achieve an improvement of up to 19.61% in perplexity and up to 27.95% improvement in GPU training time.

2 Preliminaries

2.1 Notations and Assumptions

We define a wrapper function $\texttt{Update}(S, \hat{S}) \rightarrow S$ which takes two sets, S and $\hat{S}$. It updates S by removing from it the elements in set $\hat{S}$ and returns the updated set S. In this paper, by the sum of sets (e.g., $\sum_{i=1}^{n} S_i$) we mean the concatenation of the sets which may result in a multi-set. We denote an empty set by $\emptyset$. We denote a size of vector $\boldsymbol{v}$ with $|\boldsymbol{v}|$. We assume that the server and all the users have access to secure channels among them. By the notation, $\mathcal{X} \overset{c}{\equiv} \mathcal{Y}$, we mean that the two distributions $\mathcal{X}$ and $\mathcal{Y}$ are computationally indistinguishable.

2.2 Federated Learning (FL)

The concept of FL was proposed by McMahan et al. [38] as a framework for training an ML model where the training data are distributed across multiple devices. In FL, a server orchestrates the training of a global model by aggregating models locally computed by the clients on their local devices. Suppose there are n clients, and each client D_i has a dataset S_i. The server has the initial model $\boldsymbol{\theta}$. It sends $\boldsymbol{\theta}$ to the clients, and each client performs gradient descent computation on their local dataset S_i as $J_i(S_i, \boldsymbol{\theta}) = \frac{1}{d_i} \sum_{(\mathbf{x}, y) \in S_i} C(\boldsymbol{\theta}, (\mathbf{x}, y))$, where C is the cost function and $d_i = |S_i|$. The client D_i computes the local models as $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta} - \eta \nabla J(S_i, \boldsymbol{\theta})$, where η is the learning rate. After receiving the local models from the clients, the server performs an aggregated averaging on the local models to derive the global model as:

$$\Theta = \frac{1}{d}\big(d_1\boldsymbol{\theta}_1 + \ldots + d_n\boldsymbol{\theta}_n\big) \tag{1}$$

where $d = \sum_{i=1}^{n} d_i$ is the total size of the dataset $S = \sum_{i=1}^{n} S_i$, and $\boldsymbol{\theta}_i$ is the locally trained model on the dataset S_i. This computation is repeated until the model converges.

While this framework achieves a basic level of privacy in which each user's data is not directly sent to the server, it is susceptible to advanced privacy attacks such as membership inference attacks [48,37] and data extraction attacks [14,46,18,5] as the client models θ_i are aggregated in a non-private fashion on the server side. Some privacy-preserving FL protocols have attempted to mitigate these attacks by securely aggregating the client models. These protocols rely on cryptographic techniques like homomorphic encryption [15,16], functional encryption [52], or secure aggregation [6,10]. Additionally, differentially private training techniques [3] can be used to ensure differential privacy guarantees on the global model Θ. We emphasize that in this paper, our proposed schemes are agnostic to the type of FL mechanism used.

2.3 Causal Language Modeling (CLM)

Causal Language Modeling (CLM) is a natural language processing task where the goal of the language model is to predict the next word or token given a sequence of tokens. The language model autoregressively generates the next token until a pre-determined sequence length is reached or a special STOP token is generated. Given a sequence of n tokens $\mathbf{Y} = \{y_1, y_2 \ldots, y_{n-1}, y_n\}$, the language model Θ is trained to learn the following probability distribution:

$$P(\mathbf{Y}) = \prod_{i=1}^{n} P(y_i | y_1, \ldots, y_{i-1}) \tag{2}$$

The CLM training objective is to minimize the negative log-likelihood loss given by:

$$\mathcal{L}(\Theta, \mathbf{Y}) = -\sum_{i=1}^{n} \log(\Theta(y_i | y_1, \ldots, y_{i-1})) \tag{3}$$

After training is complete, the text is autoregressively sampled from the language model i.e., $\hat{y}_{t<n} \sim \Theta(y_t | y_1, \ldots, y_{t-1})$.

The *perplexity* metric is commonly used to evaluate the performance of the language model to determine how well it has learned the probability distribution in Equation 2. The perplexity PP of a sequence $\mathbf{y}$ is defined as:

$$PP(\mathbf{Y}) = \exp\left(-\frac{1}{n}\sum_{i=1}^{n} \log(\Theta(y_i | y_1, \ldots, y_{i-1}))\right) \tag{4}$$

A lower perplexity score implies that model Θ has been trained well to estimate the real-world probability distribution. Informally, a sequence with a low perplexity score implies that the model is less "surprised" by a sequence of tokens.

2.4 Security Model

In this paper, we use the simulation-based paradigm of secure multi-party computation [21] to define and prove the proposed protocol. Since we focus on the static passive (semi-honest) adversarial model, we will restate the security definition within this context.

Two-party Computation. A two-party protocol Γ is captured by specifying a random process that maps a pair of input to a pair of outputs (one output for each party). Such process is referred to as a functionality denoted by $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^*$, where $f := (f_1, f_2)$. For every input pair (x, y), the output pair is a random variable $(f_1(x, y), f_2(x, y))$, such that the party with input x wishes to obtain $f_1(x, y)$ while the party with input y wishes to receive $f_2(x, y)$. The above functionality can be easily extended to more than two parties.

Security in the Presence of Passive Adversaries. In the passive adversarial model, the party corrupted by such an adversary correctly follows the protocol specification. Nonetheless, the adversary obtains the internal state of the corrupted party, including the transcript of all the messages received, and tries to use this to learn information that should remain private. Loosely speaking, a protocol is secure if whatever can be computed by a corrupt party in the protocol can be computed using its input and output only.

In the simulation-based model, it is required that a party's view in a protocol's execution can be simulated given only its input and output. This implies that the parties learn nothing from the protocol's execution. Formally, in two-party case, party i's view (during the execution of Γ) on input pair (x, y) is denoted by $\mathsf{View}_i^{\Gamma}(x, y)$ and equals $(w, r_i, m_1^i, ..., m_t^i)$, where $w \in \{x, y\}$ is the input of i^{th} party, r_i is the outcome of this party's internal random coin tosses, and m_j^i represents the j^{th} message this party receives. The output of the i^{th} party during the execution of Γ on (x, y) is denoted by $\mathsf{Output}_i^{\Gamma}(x, y)$ and can be generated from its own view of the execution.

Definition 1. *Let f be the deterministic functionality defined above. Protocol Γ securely computes f in the presence of a static passive probabilistic polynomial-time (PPT) adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exist PPT algorithms $(\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that:*

$$\{\mathsf{Sim}_1(x, f_1(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_1^{\mathcal{A},\Gamma}(x, y)\}_{x,y}$$

$$\{\mathsf{Sim}_2(y, f_2(x, y))\}_{x,y} \overset{c}{\equiv} \{\mathsf{View}_2^{\mathcal{A},\Gamma}(x, y)\}_{x,y}$$

Definition 1 can be easily extended to $m > 2$ parties.

2.5 Pseudorandom Function and Permutation

Informally, a pseudorandom function $\mathtt{PRF}(.)$ is a deterministic function that takes a key of length λ and an input of length u; and outputs a value of length v indistinguishable from an output of a truly random function. More formally, a pseudorandom function can be defined as $\mathtt{PRF} : \{0,1\}^{\lambda} \times \{0,1\}^{u} \rightarrow \{0,1\}^{v}$, where λ is the security parameter.

The definition of a pseudorandom permutation, $\mathtt{PRP} : \{0,1\}^{\lambda} \times \{0,1\}^{u} \rightarrow \{0,1\}^{u}$, is very similar to that of a pseudorandom function, with a difference; namely, it is required the keyed function $\mathtt{PRP}(k, \cdot)$ to be indistinguishable from a uniform permutation, instead of a uniform function. In cryptographic schemes that involve $\mathtt{PRP}$, sometimes honest parties may be required to compute the inverse of pseudorandom permutation, i.e., $\mathtt{PRP}^{-1}(k, \cdot)$, as well. In this case, it would require that $\mathtt{PRP}(k, \cdot)$ be indistinguishable from a uniform permutation even if the distinguisher is additionally given oracle access to the inverse of the permutation.

2.6 Oblivious Pseudorandom Function

An Oblivious Pseudorandom Function (OPRF) is a protocol that involves a client and a server. OPRF enables the client with input $x \in \{0,1\}^{u}$, and the server with key $k \in \{0,1\}^{\lambda}$ to execute an instance of $\mathtt{PRF}$. Informally, the security of an OPRF asserts that, by the completion of OPRF, the client only learns the output of $\mathtt{PRF}$ is evaluated on inputs k and x, i.e., $\mathtt{PRF}(k, x)$ while the server gains no information, e.g., about the input of the client and the output of $\mathtt{PRF}$.

2.7 Trusted Execution Environments

Trusted Execution Environment ($\mathcal{TEE}$), also known as a secure enclave, constitutes a secure processing environment comprising processing, memory, and storage hardware units [41,54]. Within this environment, the code and data residing in them remain isolated from other layers in the software stack, including the operating system. An ideal $\mathcal{TEE}$ guarantees the preservation of data integrity and confidentiality. Moreover, $\mathcal{TEE}$ can provide remote attestation capabilities, allowing a party to remotely verify the execution of an enclave on a genuine $\mathcal{TEE}$ hardware platform. Given the assumption that the physical CPU remains uncompromised, enclaves are shielded from attackers with physical access to the machine, including the memory and system bus. Side-channel attacks on different deployments of $\mathcal{TEE}$s have been demonstrated in the literature [49]. These attacks pose a threat as they could enable attackers to extract secrets from $\mathcal{TEE}$s. Nevertheless, $\mathcal{TEE}$s technologies have been evolving to address and mitigate side-channel attacks.

Our security, trust, and system assumptions regarding $\mathcal{TEE}$s are conservative. Specifically, as detailed in Sections 4.2 and 3.2, our solution (i) avoids disclosing any plaintext messages or private keys to $\mathcal{TEE}$ and (ii) does not expect $\mathcal{TEE}$ to maintain an extensive storage and memory space or possess strong processing resources. Instead, we establish a formal model and construction under the assumption that $\mathcal{TEE}$ guarantees execution integrity (and authenticity), and ensures minimal confidentiality. Specifically, we formally demonstrate that $\mathcal{TEE}$ at the most only learns the size of the encrypted computation result (i.e., the intersections' cardinality). In our work, $\mathcal{TEE}$ can be seamlessly substituted with any semi-honest server that does not collude with other entities.

3 Group PSI (G-PSI)

3.1 Formal Definition of G-PSI

In this section, we present the concept of Group PSI (G-PSI). In G-PSI, there are two groups of clients, $\mathcal{G}_0$ and $\mathcal{G}_1$, where each group contains m clients, $\mathcal{G}_j : \{\mathcal{C}_{j,1}, \dots, \mathcal{C}_{j,m}\}$, $0 \le j \le 1$. Each client $\mathcal{C}_{j,1}$ has a set $S_{j,1}$, such that no pair of clients' sets in the same group share a common element.

Informally, G-PSI allows every client in one group to (efficiently) find the intersection that their set has with the set of every client of the other group, without allowing them to learn anything beyond that about other clients' set elements. To achieve a high level of computational efficiency in G-PSI, we will involve a third-party $\mathcal{TP}$ that assists the clients with computing the intersection. The functionality $f_{\text{G-PSI}}$ that G-PSI computes takes a set $S_{j,i}$ from every client $\mathcal{C}_{j,i}$ and no input from $\mathcal{TP}$. It returns (i) to every client $\mathcal{C}_{j,i}$ a vector $\boldsymbol{v}_{j,i}$ which contains the intersection that $\mathcal{C}_{j,i}$'s set has with every other client's set in the other group and (ii) to $\mathcal{TP}$ an empty set $\emptyset$. Hence, functionality $f_{\text{G-PSI}}$ can be formally defined as follows,

$$f_{\text{G-PSI}}\Big(\underbrace{(S_{0,1}, \dots, S_{0,m})}_{\mathcal{G}_0}, \underbrace{(S_{1,1}, \dots, S_{1,m})}_{\mathcal{G}_1}, \emptyset\Big) \rightarrow \Big(\underbrace{(\boldsymbol{v}_{0,1}, \dots, \boldsymbol{v}_{0,m})}_{\mathcal{G}_0}, \underbrace{(\boldsymbol{v}_{1,1}, \dots, \boldsymbol{v}_{1,m})}_{\mathcal{G}_1}, \emptyset\Big) \tag{5}$$

where, $\boldsymbol{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], \dots, [S_{j,i} \cap S_{1-j,m}]\Big]$, $0 \le j \le 1$, and $1 \le i \le m$.

In the case where clients of only one of the groups, e.g., $\mathcal{G}_0$, receives the result, then the above functionality simply returns $\emptyset$ to the clients in the other group, e.g., $\boldsymbol{v}_{1,1} = \ldots = \boldsymbol{v}_{1,m} = \emptyset$.

Since $\mathcal{TP}$ performs computation on all clients' encrypted sets, there is a possibility of leakage to $\mathcal{TP}$. Depending on the protocol that realizes $f_{\text{G-PSI}}$ this leakage could contain different types of information; for instance, it could contain (i) the size of the intersection of any two clients' sets, or (ii) the size of the intersection of all clients' sets, or (iii) nothing at all. Often such leakage is defined by a leakage function $\mathcal{L}$ that takes all parties (encoded) inputs and returns the amount of leakage.

We assert that a protocol securely realizes $f_{\text{G-PSI}}$ if (1) it reveals nothing beyond a predefined leakage to $\mathcal{TP}$ and (2) whatever can be computed by a client in the protocol can be obtained from its input and output. This is formalized under the simulation paradigm. We require a client's view during an execution of G-PSI to be simulatable given its input and output. We require that the view of $\mathcal{TP}$ can be simulated given the leakage.

Definition 2 (Security of G-PSI). *Let $\mathcal{G}_0$ and $\mathcal{G}_1$ be two groups of clients, where each group contains m clients, $\mathcal{G}_j : \{\mathcal{C}_{j,1}, \ldots, \mathcal{C}_{j,m}\}$, $0 \leq j \leq 1$. Let $\mathcal{L}$ denote a leakage function, $f_{\text{G-PSI}}$ be the functionality defined above (in Relation 5 on page 6), $S = \{S_{0,1}, \ldots, S_{0,m}, S_{1,1}, \ldots, S_{1,m}\}$, and $S_{j,i}$ represent a set belonging to client $\mathcal{C}_{j,i}$. Then, a protocol Γ securely realizes $f_{\text{G-PSI}}$ in the presence of a static semi-honest PPT adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exists a PPT adversary (simulator)* Sim *in the ideal model, such that for every $\mathcal{C}_{j,i} \in \{\mathcal{C}_{0,1}, \ldots, \mathcal{C}_{0,m}, \mathcal{C}_{1,1}, \ldots, \mathcal{C}_{1,m}\}$ and $\mathcal{TP}$, Relations 6 and 7 hold respectively.*

$$\{\mathsf{Sim}_{\mathcal{C}_{j,i}}(S_{j,i}, \boldsymbol{v}_{j,i})\}_S \overset{c}{\equiv} \{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{C}_{j,i}}(S, \emptyset)\}_S \tag{6}$$

$$\{\mathsf{Sim}^{\mathcal{L}}_{\mathcal{TP}}(\emptyset, \emptyset)\}_S \overset{c}{\equiv} \{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{TP}}(S, \emptyset)\}_S \tag{7}$$

where $\boldsymbol{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], \ldots, [S_{j,i} \cap S_{1-j,m}]\Big]$, $0 \leq j \leq 1$, $1 \leq i \leq m$, and $\mathcal{TP}$'s input is $\emptyset$.

3.2 Efficient Constructions of G-PSI

In this section, we introduce two efficient protocols that realize G-PSI. The first one, called EG-PSI$^{(\text{I})}$, is highly efficient and based on symmetric key cryptography. The second one, called EG-PSI$^{(\text{II})}$, is based on OPRF (in turn depends on public key cryptography) and discloses less information to $\mathcal{TEE}$ than the former does. Thus, these two protocols trade-off between performance and leakage amount.

EG-PSI$^{(\text{I})}$. At a high level, EG-PSI$^{(\text{I})}$ operates as follows. Initially, each client in group $\mathcal{G}_0$ agrees with every client in group $\mathcal{G}_1$ on a secret key. Every client encrypts its set elements using every key it has agreed on with other clients (in a different group). Each client, for every encrypted set element, temporally stores a triple that includes (i) the encrypted element, (ii) the key used to encrypt that element, and (iii) the index of the client with whom the key was shared. These triples will enable the client to efficiently (a) retrieve the correct key and (b) identify the client with whom it has the element in common when presented with an encrypted element. Once all elements are encrypted using the corresponding keys, each client transmits only its encrypted elements to $\mathcal{TEE}$.

Given the sets of encrypted elements from all clients, $\mathcal{TEE}$ aggregates these sets and identifies the encrypted elements that appear more than once. Subsequently, $\mathcal{TEE}$ forwards to a client those encrypted elements that (1) appear more than once and (2) are among the messages that the client initially sent to $\mathcal{TEE}$. Upon receiving each encrypted element from $\mathcal{TEE}$, a client searches its local list of triples to locate the corresponding key and the index l representing a specific client. Utilizing the key, the client decrypts the element and regards the resultant element as one of the elements within the intersection it shares with the l-th client. For a comprehensive description of EG-PSI$^{(\text{I})}$, refer to Figure 1.

The only information that $\mathcal{TEE}$ learns in EG-PSI is the *size* of the intersection of any two clients' sets, called *pair-wise intersection cardinality*. Below, we formally define it.

Definition 3 (pair-wise intersection cardinality). *Let $(\underbrace{S'_{0,1}, \ldots, S'_{0,m}}_{\mathcal{G}_0})$, $(\underbrace{S'_{1,1}, \ldots, S'_{1,m}}_{\mathcal{G}_1})$ be two groups $\mathcal{G}_0$ and $\mathcal{G}_1$ of (encrypted) sets. Then, vector $\boldsymbol{s}$ represents the pair-wise intersection cardinality: $\boldsymbol{s} = [\boldsymbol{s}_{0,1}, \ldots, \boldsymbol{s}_{0,m}, \boldsymbol{s}_{1,1}, \ldots, \boldsymbol{s}_{1,m}]$, where $\boldsymbol{s}_{j,i} = \Big[\big|[S_{j,i} \cap S_{1-j,1}]\big|, \ldots, \big|[S_{j,i} \cap S_{1-j,m}]\big|\Big]$, $0 \leq j \leq 1$, and $1 \leq i \leq m$.*

Definition 4. *Let $\boldsymbol{s}$ be a pair-wise intersection cardinality of two groups $\mathcal{G}_0$ and $\mathcal{G}_1$ of encrypted sets: $(\underbrace{S'_{0,1}, \ldots, S'_{0,m}}_{\mathcal{G}_0}), (\underbrace{S'_{1,1}, \ldots, S'_{1,m}}_{\mathcal{G}_1})$, with respect to Definition 3. Then, leakage function $\mathcal{L}$ is defined as follows: $\mathcal{L}\Big((S'_{0,1}, \ldots, S'_{0,m}), (S'_{1,1}, \ldots, S'_{1,m})\Big) \rightarrow \boldsymbol{s}$.*

Theorem 1. *Let $f_{\text{G-PSI}}$ be the functionality defined in Relation 5. Also, let $\mathcal{L}$ be the leakage function presented in Definition 4. If* $\mathtt{PRP}$ *is a secure pseudorandom permutation, then EG-PSI$^{(\text{I})}$ (presented in Figure 1) securely realizes $f_{\text{G-PSI}}$, with respect to Definition 2.*

- *Parties.* Trusted execution environment $\mathcal{TEE}$, clients in group $\mathcal{G}_0 : \{\mathcal{C}_{0,1}, \ldots, \mathcal{C}_{0,m}\}$, and clients in group $\mathcal{G}_1 : \{\mathcal{C}_{1,1}, \ldots, \mathcal{C}_{1,m}\}$.
- *Inputs.* Sets $S_{0,1}, \ldots, S_{0,m}, S_{1,1}, \ldots, S_{1,m}$, where each $S_{i,j}$ belongs to client $\mathcal{C}_{i,j}$, $0 \leq j \leq 1$ and $1 \leq i \leq m$.
- *Outputs.* $\boldsymbol{v}_{j,i}$ to $\mathcal{C}_{j,i}$, where $\boldsymbol{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], \ldots, [S_{j,i} \cap S_{1-j,m}]\Big]$.

---

1. Setup.
   (a) each client $\mathcal{C}_{0,i}$ in $\mathcal{G}_0$ agrees with every client $\mathcal{C}_{1,l}$ in $\mathcal{G}_1$ on a secret key $k_{i,l}$, by picking a random key $k_{i,l}$ and sending it to $\mathcal{C}_{1,l}$. Client $\mathcal{C}_{0,i}$ stores this key as $k_{i,l}$ while $\mathcal{C}_{1,l}$ stores this key as $k_{l,i}$.
   (b) each $\mathcal{C}_{j,i}$ takes the following steps:
      i. encrypts its set elements under keys $k_{i,l}$ ($\forall l, 1 \leq l \leq m$) as follows, $\forall e \in S_{j,i} : \texttt{PRP}(k_{i,l}, e) \rightarrow e'_{i,l}$. Let set $S'_{j,i}$ contain the encrypted set elements of $\mathcal{C}_{j,i}$ and let set $T_{j,i}$ contains all triples of the form $(e'_{i,l}, k_{i,l}, l)$.
      ii. sends $S'_{j,i}$ to $\mathcal{TEE}$ and locally keeps $T_{j,i}$.
2. Finding Encrypted Intersection. $\mathcal{TEE}$ takes the following steps for each $\mathcal{C}_{j,i}$.
   (a) appends to an empty set, $R_{j,i}$, every ciphertext that satisfy the following conditions hold:
      - it appears more than once in the set $S = \sum\limits_{j=0}^{1} \sum\limits_{i=1}^{m} S'_{j,i}$.
      - it appears in set $S'_{j,i}$.
   (b) sends $R_{j,i}$ to $\mathcal{C}_{j,i}$.
3. Extracting Plaintext Intersection. Each $\mathcal{C}_{j,i}$ takes the following steps.
   (a) constructs a vector $\boldsymbol{v}_{j,i} = [\boldsymbol{v}_{j,i,1}, \ldots, \boldsymbol{v}_{j,i,m}]$, where each vector in $\boldsymbol{v}_{j,i}$ is initially empty.
   (b) decrypts each element of $R_{j,i}$ as follows. $\forall e' \in R_{j,i}$:
      i. retrieves decryption key $k_{i,l}$ and index l from $T_{j,i}$ using e'.
      ii. calls $\texttt{PRP}^{-1}(k_{i,l}, e') \rightarrow e$ and appends e to l-th vector in $\boldsymbol{v}_{j,i}$.
   (c) considers $\boldsymbol{v}_{j,i}$ as the result.

Fig. 1: First Variant of Efficient Group PSI (EG-PSI$^{(\text{I})}$).

3.3 Security Proof of EG-PSI$^{(\text{I})}$

In this section, we prove the security of EG-PSI$^{(\text{I})}$, i.e., Theorem 1.

Proof. We consider a set of cases where in each case a party is corrupt.

Corrupt Client. In this case, for the sake of simplicity, we focus on the view of a client $\mathcal{C}_{1,i}$ in $\mathcal{G}_1$. Because the clients of both groups behave the same way, with the main difference being that a client in $\mathcal{G}_0$ generates a key and sends it to each client of $\mathcal{G}_1$. Thus, the proof asserting the protocol is secure regarding a corrupt client in $\mathcal{G}_1$ will imply the security of the same protocol concerning a client in $\mathcal{G}_0$.

In the real execution of the protocol, the view of a client, $\mathcal{C}_{1,i}$, is defined as follows:

$$\{\mathsf{View}_{\mathcal{C}_{1,i}}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}(S, \emptyset)\}_S = \{S_{1,i}, r_{1,i}, \boldsymbol{k}, R_{1,i}, \boldsymbol{v}_{1,i}\}$$

where S is the set containing all clients' sets, $r_{1,i}$ is the outcome of internal random coins of $\mathcal{C}_{1,i}$, $\boldsymbol{k} = [k_{i,1}, \ldots, k_{i,m}]$ is a vector of m keys received from the clients of $\mathcal{G}_0$, $R_{1,i}$ is the encrypted intersection $\mathcal{C}_{1,i}$ received from $\mathcal{TEE}$, and $\boldsymbol{v}_{j,i}$ equals $[\boldsymbol{v}_{j,i,1}, \ldots, \boldsymbol{v}_{j,i,m}]$, each l-th vector in $\boldsymbol{v}_{j,i}$ contains the elements that are common between $\mathcal{C}_{1,i}$'s set and the set that belongs to the l-th client in the other group.

We proceed to construct a simulator, $\mathsf{Sim}_{\mathcal{C}_{1,i}}$, in the ideal model. The simulator, receiving the client's input $S_{1,i}$ and output $\boldsymbol{v}_{1,i}$, operates as follows.

1. initiates an empty view. It appends to it $S_{1,i}$ and uniformly random coins $r'_{1,i}$.

2. constructs an empty vector $\boldsymbol{k}'$ and then chooses m keys for PRP. It appends the keys to $\boldsymbol{k}'$ and adds $\boldsymbol{k}'$ to the view.
3. constructs an empty set R'. It encrypts the elements of every l-th vector $\boldsymbol{v}_{1,i,l} \in \boldsymbol{v}_{1,i}$ using l-th key k_l in $\boldsymbol{k}'$ and PRP. It inserts the ciphertexts into R'.
4. appends R' and $\boldsymbol{v}_{1,i}$ to the view. It outputs the view.

Now, we are prepared to discuss that the two views are computationally indistinguishable. The input set $S_{1,i}$ has identical distribution in both models, as their elements are identical. Furthermore, since we are in the semi-honest model, in the real execution of the protocol, the client picks its random coins $r_{1,i}$ according to the protocol description. In the ideal model, coins $r'_{1,i}$ have been selected uniformly at random. Therefore $r_{1,i}$ and $r'_{1,i}$ have identical distributions. Vectors $\boldsymbol{k}$ and $\boldsymbol{k}'$ have identical distributions too because their elements have been chosen uniformly at random. In the real model, $R_{1,i}$ contains the encrypted elements of $\boldsymbol{v}_{1,i}$, where the keys in $\boldsymbol{k}$ are used for the encryption by applying the PRP. Similarly, in the ideal model, each element of R' is the encryption of an element in $\boldsymbol{v}_{1,i}$ where a key in $\boldsymbol{k}'$ is used for the encryption with the same PRP. Thus, R and R' have identical distributions. Furthermore, elements of $\boldsymbol{v}_{1,i}$ are identical in both models; therefore, they have identical distributions in the real and ideal models.

We conclude that the views in the real and ideal models are indistinguishable.

Corrupt $\mathcal{TEE}$. In the real model, the view of $\mathcal{TEE}$ is as follows:

$$\{\mathsf{View}^{\mathcal{A},\Gamma}_{\mathcal{TP}}(S,\emptyset)\}_S = \{(S'_{0,1},\ldots,S'_{0,m}),(S'_{1,1},\ldots,S'_{1,m})\}$$

where each $S'_{j,i}$ contains the encrypted set elements of client $\mathcal{C}_{j,i}$. Next, we construct a simulator $\mathsf{Sim}^{\mathcal{L}}_{\mathcal{TEE}}$ that receives the output $\boldsymbol{s}$ of leakage function $\mathcal{L}$ (as defined in Definition 4).

1. initiates an empty view.
2. initiates m empty sets for each $\mathcal{G}_0$ and $\mathcal{G}_1$ as: $\underbrace{(S''_{0,1},\ldots,S''_{0,m})}_{\mathcal{G}_0},\underbrace{(S''_{1,1},\ldots,S''_{1,m})}_{\mathcal{G}_1}$.
3. fills each set $S''_{j,i}$ as follows, given that $\boldsymbol{s}=[\boldsymbol{s}_{0,1},\ldots,\boldsymbol{s}_{0,m},\boldsymbol{s}_{1,1},\ldots,\boldsymbol{s}_{1,m}]$, where $\boldsymbol{s}_{j,i}=\Big[\big|[S_{j,i}\cap S_{1-j,1}]\big|,\ldots,\big|[S_{j,i}\cap S_{1-j,m}]\big|\Big]$, as defined in Definition 3.
   (a) for every y-th element of each $\boldsymbol{s}_{0,i}$ (where $1\le i,y\le m$) picks $\boldsymbol{s}_{0,i}[y]$ random values (from the range of PRP). It appends these random values to both sets $S''_{0,i}$ and $S''_{1,y}$.
   (b) pads every set $S''_{j,i}$ with some random values (from the range of PRP) such that the size of each $S''_{j,i}$ after padding equals the size of $S'_{j,i}$.
4. appends sets $(S''_{0,1},\ldots,S''_{0,m}),(S''_{1,1},\ldots,S''_{1,m})$ to the view and outputs the view.

We proceed to demonstrate that the views in the real and ideal models are computationally indistinguishable. In the real model, each elements of a set $S'_{j,i}$ is an output of PRP. However, in the ideal model, each element of a set $S''_{j,i}$ has been picked uniformly at random (from the range of PRP). By the security of PRP (i.e., an output of PRP is computationally indistinguishable from a random value), sets' elements in the real and ideal model are computationally indistinguishable.

Moreover, the size of the intersection between each set $S'_{0,i}$ and each set $S'_{1,l}$ (in the real model) equals the size of the intersection between each set $S''_{0,i}$ and each set $S''_{1,l}$ (in the ideal model). Furthermore, the size of each $S'_{j,i}$ in the real model equals the size of the corresponding set $S''_{j,i}$ in the ideal model. Hence, sets $S'_{0,1},\ldots,S'_{0,m},S'_{1,1},\ldots,S'_{1,m}$ in the real model are computationally indistinguishable from sets $S''_{0,1},\ldots,S''_{0,m},S''_{1,1},\ldots,S''_{1,m}$ in the ideal model.

Thus, the views in the real and ideal models are computationally indistinguishable. $\square$

EG-PSI$^{(II)}$. Next, we present EG-PSI$^{(II)}$ which is the second variant of EG-PSI. Note that $\mathcal{TEE}$ in EG-PSI$^{(I)}$ is able to learn the cardinality of the intersection of each pair of clients' sets. We aim to reduce this leakage in EG-PSI$^{(II)}$. However, EG-PSI$^{(II)}$ is no longer based on symmetric key cryptography as it depends on OPRF. To the best of our knowledge, all the constructions of OPRFs are based on public key cryptography whose security depends on some hard problem assumptions. The same holds for EG-PSI$^{(II)}$.

A brief overview of the construction of EG-PSI$^{(II)}$ is as follows. We have two groups of clients $\mathcal{G}_0$ and $\mathcal{G}_1$ having their own set that they want to find the pairwise intersection. There is a $\mathcal{TEE}$ that has a key k for a $\mathtt{PRF}$. During the setup, each client interacts with $\mathcal{TEE}$ to encrypt their set using $\mathtt{PRF}(k, \cdot)$ through an OPRF call. Then, each client of group $\mathcal{G}_1$ will send their encrypted set to every client of $\mathcal{G}_0$. Upon receiving an encrypted set from a client of $\mathcal{G}_1$, each client of $\mathcal{G}_0$ determines the intersection with their encrypted set. Once the client finds the intersection, it marks the index of the elements in the intersection and consider the elements with the same index in the unencrypted set as elements in the intersection. Similarly, each client of $\mathcal{G}_0$ shares their encrypted set with each client of $\mathcal{G}_1$ and follows the same steps. Ultimately, every client of $\mathcal{G}_0$ knows the intersection between their set and all the sets belonging to the clients of $\mathcal{G}_1$, and the other way around. We give a detailed description of EG-PSI$^{(II)}$ in Figure 2.

- *Parties.* Trusted execution environment $\mathcal{TEE}$, clients in group $\mathcal{G}_0$: $\{\mathcal{C}_{0,1}, \ldots, \mathcal{C}_{0,m}\}$, and clients in group $\mathcal{G}_1$: $\{\mathcal{C}_{1,1}, \ldots, \mathcal{C}_{1,m}\}$.
- *Inputs.* Sets $S_{0,1}, \ldots, S_{0,m}, S_{1,1}, \ldots, S_{1,m}$, where each $S_{i,j}$ belongs to client $\mathcal{C}_{i,j}$, $0 \le j \le 1$ and $1 \le i \le m$.
- *Outputs.* $\boldsymbol{v}_{j,i}$ to $\mathcal{C}_{j,i}$, where $\boldsymbol{v}_{j,i} = \Big[[S_{j,i} \cap S_{1-j,1}], \ldots, [S_{j,i} \cap S_{1-j,m}]\Big]$.

---

1. *Setup.*
   $\mathcal{TEE}$ generates a secret key k for $\mathtt{PRF}$ that will be used for an OPRF evaluation.
2. *Encryption of each client's set.*
   Each client $\mathcal{C}_{j,i}$ and $\mathcal{TEE}$ run an OPRF protocol and obtains the encrypted set $\mathtt{PRF}(k, S_{j,i})$, where $0 \le j \le 1$ and $1 \le i \le m$.
3. *Finding Encrypted Intersections.*
   (a) Each client $\mathcal{C}_{j,i}$ sends each $\mathtt{PRF}(k, S_{j,i})$ to every client $\mathcal{C}_{1-j,i}$ (in the other group), where $0 \le j \le 1$ and $1 \le i \le m$.
   (b) Each client $\mathcal{C}_{j,i}$, takes the following steps.
      i. Creates an empty set $R_{j,i,t}$ for all $1 \le t \le m$.
      ii. Let e_ℓ represents an element of $\mathtt{PRF}(k, S_{j,i})$, where $1 \le \ell \le |\mathtt{PRF}(k, S_{j,i})|$.
      Performs $R_{j,i,t} \leftarrow R_{j,i,t} \cup \{\ell\}$ if e_ℓ appears both in $\mathtt{PRF}(k, S_{j,i})$ and $\mathtt{PRF}(k, S_{j-1,i})$.
4. *Extracting Plaintext Intersection.*
   Each $\mathcal{C}_{j,i}$ for $j = 0, 1$, and $1 \le i \le m$ does the following.
   (a) Constructs a vector $\boldsymbol{v}_{j,i} = [\boldsymbol{v}_{j,i,1}, \ldots, \boldsymbol{v}_{j,i,m}]$, where each vector in $\boldsymbol{v}_{j,i}$ is initially empty.
   (b) Appends the ℓ-th element of $S_{j,i}$ to $\boldsymbol{v}_{j,i,t}$ for all $\ell \in R_{j,i,t}$.
   (c) Returns $\boldsymbol{v}_{j,i}$.

Fig. 2: Second Variant of Efficient Group PSI (EG-PSI$^{(II)}$).

In EG-PSI$^{(II)}$, $\mathcal{TEE}$ does not participate in the protocol except the OPRF evaluation and that is also a one-time activity. As a result, $\mathcal{TEE}$ does not learn anything about the set intersection, it only learns the cardinality of each client's set.

Theorem 2. *Let $f_{\text{G-PSI}}$ be the functionality defined in Relation 5 (on page 6). Also, let $\mathcal{L}$ be the leakage function presented in Definition 4 with the empty output. If OPRF is secure, then EG-PSI$^{(II)}$ securely realizes $f_{\text{G-PSI}}$, w.r.t. Definition 2.*

3.4 Security Proof of EG-PSI$^{(II)}$

Proof (sketch). It is not hard to simulate parties' views. $\mathcal{TEE}$'s view in the real model can be easily simulated given that the main information it learns within the protocol execution includes its view during the invocation of the OPRF instances (but not the outputs of OPRF). If OPRF is secure, then the views of $\mathcal{TEE}$ within the real and ideal model will be computationally indistinguishable in this protocol.

Each client's view (in the real execution of the protocol) mainly includes the outputs of OPRF received from its counterparts and the exchanged messages during OPRF's execution with $\mathcal{TEE}$. Due to the security of OPRF, an output of OPRF is computationally indistinguishable from a value picked uniformly at random from the output range of OPRF. Also, the messages each client receives during the execution of OPRF can be simulated, due to the security of OPRF. $\square$

Note that both variants remain secure in the presence of semi-honest adversaries. However, if $\mathcal{TEE}$ or a client is a malicious (or active) adversary, then it can affect the correctness of the result, without being detected.

4 Privacy-Preserving Multi-Party Deduplication (P-MPD)

4.1 Formal Definition of P-MPD

P-MPD considers the setting where there are n clients $C = \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$ and each $\mathcal{C}_i \in C$ has a set S_i. P-MPD enables the clients to (efficiently) remove duplicates from their local sets such that after the deduplication the concatenation of all clients' local sets equals the union of their initial sets. The functionality $f_{\text{P-MPD}}$ that P-MPD computes takes a set S_i from each $\mathcal{C}_i$. It returns an updated (i.e., deduplicated) set S'_i to each $\mathcal{C}_i$. Formerly, $f_{\text{P-MPD}}$ can be defined as follows:

$$f_{\text{P-MPD}}(S_1, \ldots, S_m) \rightarrow (S'_1, \ldots, S'_m) \tag{8}$$

where $\sum\limits_{i=1}^{m} S'_i = \bigcup\limits_{i=1}^{m} S_i = S_{\cup}$ and $S_{\cup}$ denotes the union of all initial sets and contains only unique elements.

To maintain generality, we define a leakage function, denoted as $\mathcal{W}$. The output of this function relies on the protocol implementing $f_{\text{P-MPD}}$. We assert that a protocol securely realizes $f_{\text{P-MPD}}$ if whatever can be computed by a party in the protocol can be derived solely from its individual input and output, given the output of $\mathcal{W}$. Below, we formally state it.

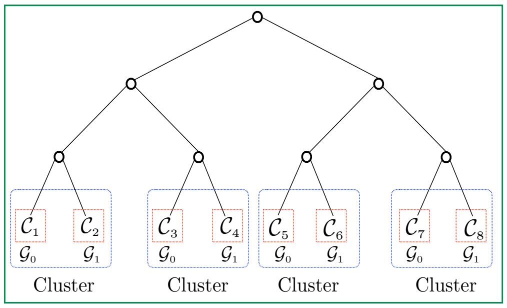
Definition 5 (Security of P-MPD). *Let $S = \{S_1, \ldots, S_m\}$, and S_i represent a set belonging to client $\mathcal{C}_i$. Let $\mathcal{W}$ denote a leakage function and $f_{\text{P-MPD}}$ be the functionality defined in Relation 8. Then, a protocol Ψ securely realizes $f_{\text{P-MPD}}$ in the presence of a static semi-honest PPT adversary $\mathcal{A}$, if for every $\mathcal{A}$ in the real model, there exists a PPT* Sim *in the ideal model, such that for every $\mathcal{C}_i \in \{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$, Relation 9 holds.*

$$\{\mathsf{Sim}^{\mathcal{W}}_{\mathcal{C}_i}(S_i, S'_i)\}_S \stackrel{c}{\equiv} \{\mathsf{View}^{\mathcal{A},\Psi}_{\mathcal{C}_i}(S)\}_S \tag{9}$$

4.2 Constructions of P-MPD

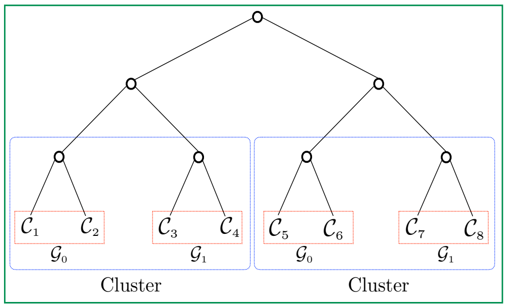
Construction Based on EG-PSI. We introduce a pioneering protocol called Efficient Privacy-Preserving Multi-Party Deduplication (EP-MPD) that realizes P-MPD efficiently. The idea behind EP-MPD's design involves constructing a binary tree with leaf nodes representing the clients' indices. At each level, each cluster is formed with two different groups of clients, namely $\mathcal{G}_0$ and $\mathcal{G}_1$. Subsequently, EG-PSI is recursively applied to the sets of clients sharing the same cluster until the tree's root is reached. After each invocation of EG-PSI, the clients of $\mathcal{G}_0$ update their sets by removing the intersections, returned by EG-PSI, from their local sets. These updated sets are then used as input in the subsequent EG-PSI invocation.

Next, we delve into further detail. We begin by sorting the clients' indices in ascending order. Next, we construct a binary tree such that the leaf nodes represent the clients' indices. At level $d = 1$, commencing from the left-hand side, every two clients form a cluster. Within each cluster, the first client is assigned to group $\mathcal{G}_0$ and the second client to group $\mathcal{G}_1$. Then, the clients within the same cluster engage in an instance of EG-PSI. Upon the return of the sets' intersection by EG-PSI, only the client situated on the left-hand side, specifically the one with the smaller index, modifies its local set by eliminating the elements of the intersection from its sets. This process is outlined in Figure 3a.



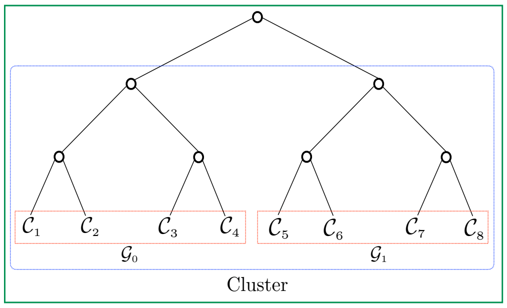
$\mathcal{C}_1$ $\mathcal{C}_2$ $\mathcal{C}_3$ $\mathcal{C}_4$ $\mathcal{C}_5$ $\mathcal{C}_6$ $\mathcal{C}_7$ $\mathcal{C}_8$
$\mathcal{G}_0$ $\mathcal{G}_1$
Cluster

(a) Demonstration of clients joining various groups and clusters at level 1.



$\mathcal{C}_1$ $\mathcal{C}_2$ $\mathcal{C}_3$ $\mathcal{C}_4$ $\mathcal{C}_5$ $\mathcal{C}_6$ $\mathcal{C}_7$ $\mathcal{C}_8$
$\mathcal{G}_0$ $\mathcal{G}_1$
Cluster

(b) Illustration of clients joining various groups and clusters at level 2.



$\mathcal{C}_1$ $\mathcal{C}_2$ $\mathcal{C}_3$ $\mathcal{C}_4$ $\mathcal{C}_5$ $\mathcal{C}_6$ $\mathcal{C}_7$ $\mathcal{C}_8$
$\mathcal{G}_0$ $\mathcal{G}_1$
Cluster

(c) Depiction of how clients become part of diverse groups and clusters at level 3.

Fig. 3: The process of eight clients forming clusters and groups at various levels of a binary tree. At each level, clients belonging to groups $\mathcal{G}_0$ and $\mathcal{G}_1$ within the same cluster initiate EG-PSI, followed by the updating of their respective local sets.

At level $d = 2$, starting from the left-hand side, every set of 2^d clients forms a cluster. Within each cluster, the initial $\frac{2^d}{2} = 2$ clients enter group $\mathcal{G}_0$ and the remainder (i.e., the remaining 2 clients) enter group $\mathcal{G}_1$. Subsequently, the clients within the same cluster engage in an instance of EG-PSI. Once again, upon the return of their sets' intersection by EG-PSI, the clients possessing the smaller indices update their local sets. This procedure is illustrated in Figure 3b. The process continues until level $d = \log_2 m$ is reached. At this point, all m clients collectively engage in an instance of EG-PSI. Each $\frac{2^d}{2} = \frac{m}{2}$ clients enter group $\mathcal{G}_0$ and the remaining $\frac{m}{2}$ clients enter group $\mathcal{G}_1$. These m clients jointly participate in an instance of EG-PSI. Similarly to previous steps, the clients with the smaller indices update their local sets based on the output of EG-PSI. This procedure is outlined in Figure 3c.

– *Parties.* A set of clients $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$.
– *Inputs.* Sets $S_1, \ldots, S_m$, where each S_i belongs to client $\mathcal{C}_i$, $1 \leq i \leq m$, and m is a power of 2.
– *Outputs.* Updated sets $S'_1, \ldots, S'_m$, where each S'_i belongs to client $\mathcal{C}_i$, and $\sum\limits_{i=1}^{m} S'_i = \bigcup\limits_{i=1}^{m} S_i$.

The clients take the following steps $\forall d$, $1 \leq d \leq \log_2(m)$:

1. *Forming Clusters.* Every distinct 2^d clients enter the same cluster. Thus, for every level d, there will be $\frac{m}{2^d}$ cluster(s). For instance, when $d = 1$, then there are the following clusters: $(\mathcal{C}_1, \mathcal{C}_2), \ldots, (\mathcal{C}_{m-1}, \mathcal{C}_m)$.
2. *Forming Group.* In each cluster, the first $\frac{2^d}{2}$ clients enter group 0, $\mathcal{G}_0$, and the rest of the clients enter group 1, $\mathcal{G}_1$.
3. *Finding Intersection.* In each cluster, the clients of $\mathcal{G}_0$ and $\mathcal{G}_1$ together engage an instance of EG-PSI. Each client $\mathcal{C}_j$ receives a vector $\boldsymbol{v}_j$ from EG-PSI, where $\boldsymbol{v}_j$ specifies the elements that $\mathcal{C}_j$ has in common with each client of the other group in the same cluster. It would be sufficient if only clients of $\mathcal{G}_0$ in each cluster were aware of the intersection result.
4. *Removing Duplication.* For every element $e \in \boldsymbol{v}_j$, each $\mathcal{C}_j$ in $\mathcal{G}_0$ calls $\texttt{Update}(S_j, \{e\}) \rightarrow S_j$, where $S_j \leftarrow S_j \setminus \{e\}$.

Fig. 4: Efficient Privacy-Preserving Multi-Party Deduplication (EP-MPD): the construction of P-MPD.

Figure 4 presents a detailed description of EP-MPD. For the sake of simplicity, we assume that m is a power of 2 in EP-MPD's description. However, we do not impose on the number of clients. Non-powers of 2 will result in incomplete sub-trees in the formation of clusters in the DE protocol where each group may have an unequal number of clients. This does not affect the EP-MPD protocol execution.

4.3 Naive Approaches

Running a Two-Party PSI Multiple Times. One might be tempted to allow each client to invoke an existing two-party PSI with every other client. However, this approach in total requires $\binom{m}{2} = O(m^2)$ invocations of the two-party PSI. Whereas EP-MPD requires only $\sum\limits_{i=1}^{\log_2(m)} (\frac{m}{2^i}) = m - 1$ invocations of our efficient tailor-made PSI, EG-PSI. To provide concrete values, when $m = 256$, the naive scheme requires 32,640 invocations of a standard two-party PSI, whereas our scheme requires only 255 invocations of EG-PSI.

Running Multi-Party PSI Once. One might be tempted to execute an existing multi-party PSI protocol only once on all clients' sets, hoping to identify and subsequently remove duplicated elements. Nevertheless, this approach falls short of identifying all duplications. The limitation arises from the fact that a multi-party PSI can uncover elements common to *all* clients, making it incapable of detecting elements shared among only a subset of clients.

Deduplicating all Sets at Once. There is a simplified version of EP-MPD. In this variant, each client reaches an agreement with the others on a secret key. Subsequently, each client encrypts its set elements using every key agreed upon with the other clients. Finally, all clients transmit their encrypted set elements to $\mathcal{TEE}$. Upon receiving the encrypted sets from all clients, $\mathcal{TEE}$ identifies all duplicated elements. It then transmits to each client: (1) the duplicated elements found in the original elements that the respective client sent to $\mathcal{TEE}$, and (2) the index of the client with which it shares the same element. Based on $\mathcal{TEE}$'s response, the clients with smaller indices update their local sets accordingly.

However, this approach requires that $\mathcal{TEE}$ maintains a local storage or memory space equal to the size of the concatenation of all encrypted sets. In contrast, EP-MPD with the underlying EG-PSI building block, allows $\mathcal{TEE}$ to have a substantially smaller storage or memory space. This is achieved because, before $\mathcal{TEE}$ receives all clients' encrypted sets at level $d = log_2(m)$, clients apply updates to their sets multiple times.

This iterative process progressively reduces the size of the clients' sets, and the reduction can be particularly significant when the number of duplications is substantial.

4.4 Security Proof of EP-MPD

In any secure (multi-party) deduplication protocol, a participating client will learn the exact redundant elements within its set upon the protocol's completion. However, in EP-MPD, a client gains slightly more information. To be specific, upon the completion of EP-MPD, each client acquires the knowledge of the index (or identity) of the client that shares the redundant element. This additional information gained by a client during the execution of EP-MPD is formally defined as the output of a leakage function $\mathcal{W}$ which we define below.

$\mathcal{W}$ takes as input (i) sets $S_1, \ldots, S_m$, where each set S_i belongs to client $\mathcal{C}_i$, and (ii) sets $R_1, \ldots, R_m$, where each R_i contains the redundancy (or intersection) that a set S_i has with every other set. $\mathcal{W}$ outputs to the i-th client a set Q_i of triples, where each triple is of the form $(r_{i,l}, i, l)$, $r_{i,l} \in R_i$ is a redundancy between sets S_i and S_l.

Definition 6. *Let $S = \{S_1, \ldots, S_m\}$ be a set of sets, where each set S_i belongs to client $\mathcal{C}_i$. Also, let $R = \{R_1, \ldots, R_m\}$ be a set of sets, where each set R_i comprises the redundancy that S_i has with every other j-th set, for all j, $1 \leq j \leq m$ and $i \neq j$. Then, leakage function $\mathcal{W}$ is defined as follows: $\mathcal{W}(S, R) \rightarrow (Q_1, \ldots, Q_m)$, where each Q_i is a set of triples of the form $(r_{i,l}, i, l)$, $r_{i,l} \in R_i$ is a redundancy between sets S_i and S_l.*

Theorem 3. *Let $f_{\text{P-MPD}}$ be the functionality defined in Relation 8 and $\mathcal{W}$ be the leakage function presented in Definition 6. If EG-PSI is secure (w.r.t. Definition 2), then EP-MPD (presented in Figure 4) securely realizes $f_{\text{P-MPD}}$, w.r.t. Definition 5.*

We proceed to prove the security of EP-MPD, i.e., Theorem 3. For the sake of concreteness and because we have provided a detailed proof for EG-PSI$^{(\text{I})}$, we will prove Theorem 3 when EP-MPD relies on EG-PSI$^{(\text{I})}$. The proof for EG-PSI$^{(\text{I})}$-based EP-MPD will easily follow.

Proof. In the real execution of the protocol, the view of a client, $\mathcal{C}_i$, is defined as follows:

$$\{\mathsf{View}_{\mathcal{C}_i}^{\mathcal{A},\text{EP-MPD}}(S)\}_S = \{S_i, \mathbf{View}_{\mathcal{C}_i}, \boldsymbol{v}_i\}$$

where $S = \{S_1, \ldots, S_m\}$, S_i represents a set belonging to $\mathcal{C}_i$, $\mathbf{View}_{\mathcal{C}_i} = [\mathsf{View}_{\mathcal{C}_i,1}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}, \ldots, \mathsf{View}_{\mathcal{C}_i,z}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}]$ contains $z = log_2(m)$ views of $\mathcal{C}_i$ such that each view $\mathsf{View}_{\mathcal{C}_i,d}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}$ corresponds to $\mathcal{C}_i$'s view when it invokes EG-PSI$^{(\text{I})}$ at level d, where $1 \leq d \leq z$. Moreover, $\boldsymbol{v}_i$ contains z vectors, where each vector $\boldsymbol{v}_{i,d}$ in $\boldsymbol{v}_i$ is the output of EG-PSI$^{(\text{I})}$ that client $\mathcal{C}_i$ receives at level d. We proceed to construct a simulator, $\mathsf{Sim}_{\mathcal{C}_i}$, in the ideal model. The simulator, receives the output Q_i of leakage function $\mathcal{W}$ (as defined in Definition 6), the client's input S_i and output $\boldsymbol{v}_i$. $\mathsf{Sim}_{\mathcal{C}_i}$ operates as follows.

1. initiates an empty view.
2. for each level d (where $1 \leq d \leq z$) constructs the following set for each client $\mathcal{C}_l$ in the other group, using (i) the result $\boldsymbol{v}_{i,d}$ that $\mathcal{C}_i$ receives at level d and (ii) the leakage function's output Q_i (hence in total, in level d, the simulator will generate $\frac{2^d}{2}$ sets on behalf of the clients in the other group).
   (a) constructs an empty set S_l.
   (b) finds every triple (x, a, b) in Q_i, such that $a = i$ and $b = l$. It adds the first element x of each triple that meets the above conditions to S_l. This ensures that S_l contains all the elements in common with S_i.
   (c) pads every set S_l with some random values (from the set universe) such that the size of each S_l, in the ideal model, after padding equals the size of S_l in the real model.
3. generates $\frac{2^d}{2} - 1$ empty sets for the group to which S_i belongs. It appends to these empty sets values picked uniformly at random from the set universe. It ensures the size of each of these sets in the ideal model equals to that of in the real model.

4. for every level d:
   (a) invokes an instance of EG-PSI$^{(\text{I})}$ using the following sets: (i) S_i and sets generated in step 3, that are placed in one group and (ii) sets generated in step 2, placed in another group. As a result, it receives output $\boldsymbol{v}_{i,d}$.
   (b) extracts $\mathcal{C}_i$'s view $\mathsf{View}_{\mathcal{C}_i,d}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}$ after the execution of EG-PSI$^{(\text{I})}$.
   (c) appends every $\mathsf{View}_{\mathcal{C}_i,d}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}$ and $\boldsymbol{v}_{i,d}$ to its view.
   (d) outputs the view.

Now, we are prepared to show that the two views (in the real and ideal models) are computationally indistinguishable. The input S_i in both models are identical, therefore they will have identical distribution. Due to the security of EG-PSI$^{(\text{I})}$, for every level d, $\mathsf{View}_{\mathcal{C}_i,d}^{\mathcal{A},\text{EG-PSI}^{(\text{I})}}$ in the real and ideal models are computationally indistinguishable, as demonstrated in the proof of Theorem 1. Furthermore, the intersection $\boldsymbol{v}_{i,d}$ that $\mathcal{A}$ learns at every level d in the real and ideal model are identical, thus they are indistinguishable.

Hence, the views of $\mathcal{A}$ in the real and ideal models are computationally indistinguishable. □

5 Enhanced FL: Applying EP-MPD to FL

With all essential building blocks in place, we are prepared to elucidate the complete functionality of the system. There are clients $\mathcal{C}_1, \ldots, \mathcal{C}_m$ having the sets $S_1, \ldots, S_m$, where each S_i belongs to client $\mathcal{C}_i$ respectively. Phase 1 involves each client conducting local deduplication. Since the local deduplication is a private computation there is no privacy requirement. The data owners can apply deduplication techniques as illustrated in [31] which results in having a set S'_i free of duplicates by each client $\mathcal{C}_i$. Subsequently, in Phase 2 they employ EP-MPD to eliminate duplicates across all clients. At the end of this phase, client $\mathcal{C}_i$ gets an updated set S''_i, such that $\sum_{i=1}^{m} S''_i = \bigcup_{i=1}^{m} S'_i$. Moving to Phase 3, they adopt a FL protocol (outlined in Section 2.2) to train the model. Further details of this procedure are provided in Figure 5.

– *Parties.* A set of clients $\{\mathcal{C}_1, \ldots, \mathcal{C}_m\}$.
– *Server.* Holds the initial model $\boldsymbol{\theta}$.
– *Inputs.* Sets $S_1, \ldots, S_m$, where each S_i belongs to client $\mathcal{C}_i$, $1 \le i \le m$, and m is a power of two.
– *Outputs.* A global updated model Θ.

1. *Local Deduplication.* Each client runs a deduplication algorithm on their local dataset. At the end, client $\mathcal{C}_i$ receives an updated dataset S'_i.
2. *Global Deduplication.*
   (a) All the clients participate in the EP-MPD as described in Figure 4.
   (b) Each client $\mathcal{C}_i$ gets updated set S''_i, such that
$$\sum_{i=1}^{m} S''_i = \bigcup_{i=1}^{m} S'_i.$$
3. *Federated learning.*
   (a) The server and clients agree upon an FL protocol for training.
   (b) The server initiates the learning by sharing the initial model $\boldsymbol{\theta}$ with each client.
   (c) Each client $\mathcal{C}_i$ trains on their local dataset S''_i and updates $\boldsymbol{\theta}$ to $\boldsymbol{\theta}_i$.
   (d) The clients and server aggregate the local models $\boldsymbol{\theta}_i$ trained by the clients.
   (e) The server outputs the global updated model Θ for the next training round.

Fig. 5: Federated learning with deduplication.

6 Experiments

6.1 Implementation Details

We implement the EP-MPD protocol in `python=3.10`. We use AES with a 128-bit key in CBC mode from the `cryptography` library as a PRP in EG-PSI$^{(\mathrm{I})}$. To realize an OPRF in EG-PSI$^{(\mathrm{II})}$, we use the OPRF algorithm proposed in [12] from the `oprf` library. All EP-MPD experiments are conducted on a batch computing facility with a Dual Intel Xeon E5-2660 v3 @ 2.6 GHz CPU and 60 GB RAM. For the FL experiments, we implement a FL setting for CLMs using `python=3.10` with the `transformers=4.40.1` and `torch=2.3` libraries. We implement the FedAvg [38] algorithm for FL training to analyze the effect of EP-MPD on FL's performance. However, in practice, we recommend combining EP-MPD with privacy-preserving FL protocols for end-to-end privacy guarantees. All FL experiments are conducted on a server with an Intel(R) Xeon(R) Gold 6336Y CPU @ 2.40GHz CPU and NVIDIA RTX A6000 GPU.

We run all clients sequentially on a machine for both experiments. However, to emulate a real-world setting where clients run on different devices at the same time, we first compute the individual client running times and then estimate the real-world distributed running time by assuming the clients have run simultaneously. We open-source our implementations[4].

While implementing EP-MPD, we have two options for the EG-PSI module: EG-PSI$^{(\mathrm{I})}$ and EG-PSI$^{(\mathrm{II})}$. We observe that EG-PSI$^{(\mathrm{II})}$-based EP-MPD calls an OPRF function for every client in the leaf nodes of the EP-MPD tree, returning an encryption of the clients' sets. As clients move to upper levels, their set elements remain unchanged, with only the cardinality potentially changing. Therefore, except at the bottom level, OPRF calls are unnecessary for encryption, allowing the same encrypted set or their subsets to be reused at other levels. Thus, we have slightly adjusted the implementation. This significantly reduces time without compromising security properties, as reflected in our experiments.

6.2 EP-MPD Benchmarks

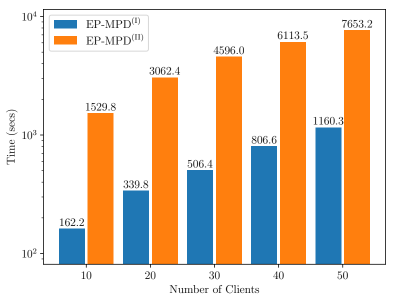
Experimental Setup. We designed three experiment settings to benchmark the performance of EP-MPD. The three settings analyze the effect of the number of clients, dataset size, and duplication percentage on the runtime of EP-MPD. All experiments were performed with both EG-PSI$^{(\mathrm{I})}$ and EG-PSI$^{(\mathrm{II})}$ as building blocks. The duplication percentage refers to the percentage of samples in a client's dataset present in another client's dataset. The three experiment settings are:

a. The dataset size is fixed at $|S_i| = 2^{19}$, the duplication percentage at 30%, while the number of clients m is varied. Specifically, m is set to each value in $\{10, 20, 30, 40, 50\}$.
b. The number of clients is fixed at 50 and the duplication percentage at 30%, while the dataset size $|S_i|$ is varied. Specifically, $|S_i|$ is set to each value in $\{2^{10}, 2^{11}, 2^{13}, 2^{15}, 2^{17}, 2^{19}\}$.
c. The database size $|S_i|$ is fixed at 2^{19}, the number of clients at 50, while the duplication percentage is varied. Specifically, it is set to each value in $\{10\%, 30\%, 50\%, 70\%, 90\%\}$.

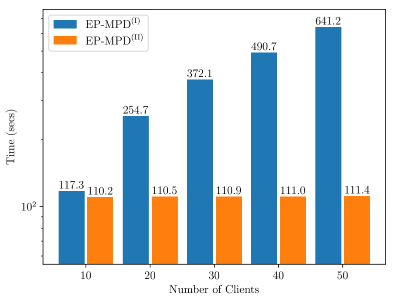
We create pairwise duplicates between clients such that every client has a unique duplicate element with every other client. All set elements are 32-bit integers. We assume that $\mathcal{TEE}$ can receive data in parallel from the clients. All other interactions that require computation between $\mathcal{TEE}$ and clients occur sequentially. In the remainder of this section, by EP-MPD$^{(\mathrm{I})}$ and EP-MPD$^{(\mathrm{II})}$ we mean the EP-MPD that uses EG-PSI$^{(\mathrm{I})}$ and EG-PSI$^{(\mathrm{II})}$ respectively.

Effect of Client Count. Figure 6 depicts the effect of the number of clients on (a) EP-MPD running time, (b) the average running time of clients, and (c) $\mathcal{TEE}$ running time. Figure 6a demonstrates that the EP-MPD running time increases linearly with the number of clients for both EP-MPD$^{(\mathrm{I})}$ and EP-MPD$^{(\mathrm{II})}$. The increase in the client count increases the number of PRP and OPRF evaluations as the total number of elements across all clients grows. We observe that EP-MPD$^{(\mathrm{I})}$ is about 7–10× faster than EP-MPD$^{(\mathrm{II})}$.

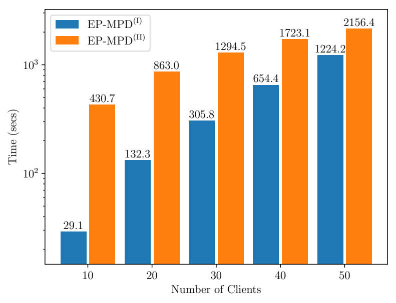
[4] `https://github.com/vdasu/deduplication`



(a) EP-MPD running time.



(b) Client running time.



(c) $\mathcal{TEE}$ running time.

Fig. 6: Effect of client count with 2^{19} dataset size and 30% duplication percentage on running time (logarithmic scale).

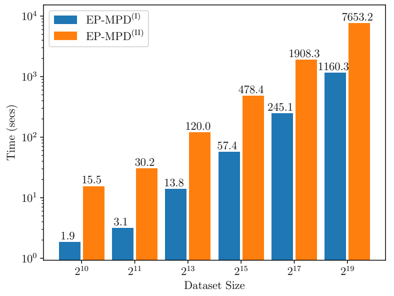
The average running time of the client however shows a different trend. The client running time in EP-MPD$^{(I)}$ increases linearly with the client count and is always greater than – up to 6× – the running time of EP-MPD$^{(II)}$. The running time of EP-MPD$^{(II)}$ is constant as the number of clients increases. In EP-MPD$^{(I)}$, the client-side running time increases with the number of clients, because each client has to perform more PRP evaluations. However, in EP-MPD$^{(II)}$, the number of OPRF evaluations is equal to the number of elements in each client's set. Additionally, in EP-MPD$^{(I)}$, the clients perform the PRP evaluations at every level of the tree. However, in EP-MPD$^{(II)}$ the OPRF evaluations are only performed once when the clients are at the leaf level. The running time of $\mathcal{TEE}$ follows a similar trend to the EP-MPD running time. The $\mathcal{TEE}$ running time in EP-MPD$^{(II)}$ is always higher than EP-MPD$^{(I)}$. Both exhibit linear increases in running time as the number of clients increases. The running time in EP-MPD$^{(II)}$ is much higher as the OPRF evaluations are relatively expensive.

Effect of Dataset Size. Figure 7 illustrates the effect of dataset size on EP-MPD running time, the average running time of clients, and $\mathcal{TEE}$ running time. From Figure 7a, it is evident that the EP-MPD running time grows linearly with increasing the dataset size. EP-MPD$^{(I)}$ is up to 8× faster than EP-MPD$^{(II)}$. As the dataset size increases, the number of PRP and OPRF evaluations grows, thereby increasing the EP-MPD running time. Figure 7b shows the effect of dataset size on client running time. Unlike increasing the client count, increasing the dataset size increases the client running time of both EP-MPD$^{(I)}$ and EP-MPD$^{(II)}$. We observe this difference because an increase in dataset size increases the number of OPRF evaluations while any change in the client count does not. The client running time of EP-MPD$^{(II)}$ is up to 7× faster than EP-MPD$^{(I)}$.

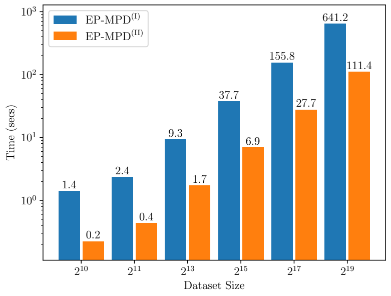
Figure 7c depicts the effect of dataset size on the running time of $\mathcal{TEE}$. The growth of dataset size increases the OPRF evaluations in EP-MPD$^{(II)}$. In EP-MPD$^{(I)}$, $\mathcal{TEE}$ has to identify more overlapping ciphertexts as the datasets are larger. The running time of $\mathcal{TEE}$ increases linearly with the dataset size for both cases. $\mathcal{TEE}$ running time in EP-MPD$^{(I)}$ is up to 5× faster than EP-MPD$^{(II)}$.

Effect of Duplication Percentage. Figure 8 shows the effect of duplication percentage on EP-MPD running time, the average running time of clients, and $\mathcal{TEE}$ running time. Figure 8a highlights the running time of EP-MPD as the duplication percentage changes. We note that in EP-MPD$^{(II)}$, OPRF evaluations incur the most overwhelming running time, dominating all other running time costs. Overall, the running time of EP-MPD$^{(II)}$ is constant while that of EP-MPD$^{(I)}$ linearly decreases as the duplication rate grows. In EP-MPD$^{(II)}$, the number of OPRF evaluations is independent of the duplication percentage, because the OPRF is always performed only when the clients are at the leaf level. However, in EP-MPD$^{(I)}$, the number of PRP invocations decreases as clients move up the tree from the leaf level because a higher duplication percentage removes more elements in each level of the tree. The total running time of EP-MPD$^{(I)}$ is up to 7× faster than EP-MPD$^{(II)}$.

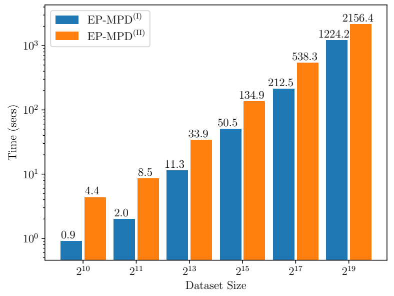
Furthermore, the client-side running time of EP-MPD$^{(II)}$ is constant while that of EP-MPD$^{(I)}$ linearly decreases as the duplication percentage increases, as shown in Figure 8b. The reason is identical to the



(a) EP-MPD running time.

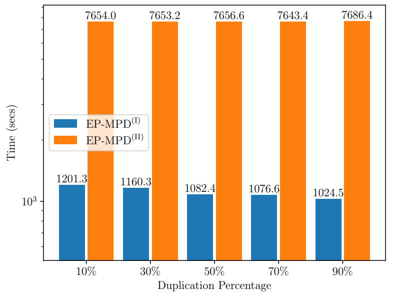


(b) Client running time.

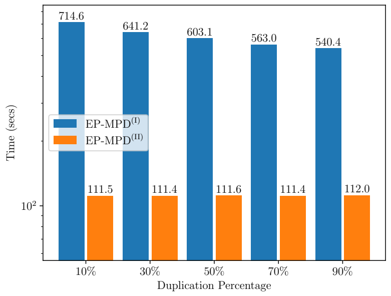


(c) $\mathcal{TEE}$ running time.

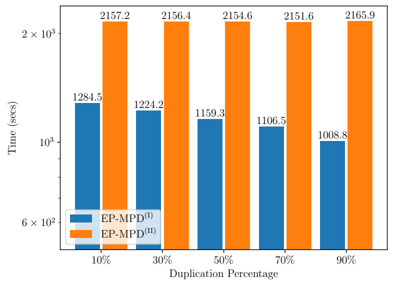
Fig. 7: Effect of dataset size with 50 clients and 30% duplication percentage on running time (logarithmic scale).



(a) EP-MPD running time.



(b) Client running time.



(c) $\mathcal{TEE}$ running time.

Fig. 8: Effect of duplication percentage with 2^{19} dataset size and 50 clients on running time (logarithmic scale).

observation of the EP-MPD running time. The number of PRP evaluations reduces while the number of OPRF evaluations does not. Similar to the prior experiment settings, the client time of EP-MPD$^{(I)}$ is higher than EP-MPD$^{(II)}$ and is up to 7× slower. Figure 8c shows the effect of duplication percentage on the running time of $\mathcal{TEE}$. The $\mathcal{TEE}$ running time in EP-MPD$^{(II)}$ is constant while in EP-MPD$^{(I)}$ linearly decreases. The $\mathcal{TEE}$ running time in EP-MPD$^{(I)}$ is up to 2× faster than EP-MPD$^{(II)}$. The overall running time of the symmetric key-based EP-MPD$^{(I)}$ is considerably less than the public key-based EP-MPD$^{(II)}$. Nevertheless, the clients in EP-MPD$^{(I)}$ need to perform more computations compared to those in EP-MPD$^{(II)}$.

6.3 FL with EP-MPD

Experimental Setup. We create an FL setting with varying levels of duplication to analyze the effect of EP-MPD on the model performance and training efficiency of CLM fine-tuning. We use GPT-2 Medium and GPT-2 Large CLMs from the GPT-2 [43] family of decoder-only models based on the Transformer [50] architecture. The GPT-2 Medium has 345 million parameters with 24 decoder blocks, 16 attention heads, and 1024 hidden sizes while GPT-2 Large has 774 million parameters with 36 decoder blocks, 20 attention heads, and 1280 hidden sizes. Both models have a sequence length of 1024 tokens.

We create 10 clients and experiment with the following 7 datasets. The Rotten Tomatoes and IMDB datasets contain movie reviews. The Sonnets and Plays datasets are a collection of works by William Shakespeare. Poetry is a collection of short poems by a variety of poets. Haiku and Short Jokes are a collection of short Japanese-style poems and Jokes from Reddit respectively. The language modeling task for each dataset is to learn how to generate text from the dataset's distribution. For all datasets, we create a Train and Test split in an 80:20 ratio. Model perplexity is evaluated on the Test set after training. We follow a two-step procedure to create the client datasets with $x\%$ duplication. Initially, we equally distributed the Train set among all 10 clients. Next, we sample with replacement $x\%$ of the original Train set. This subset is then again equally distributed among all clients. We perform 5 rounds of FL training with at most 10 epochs of local training. The GPT-2 Large models were trained for fewer epochs and with less data because they were overfitting to the Train set for more epochs and data. Additional details about the datasets and FL training hyperparameters can be found in Appendix B.

FL Model Performance. We measure the perplexity of the final model on the Test set to assess the impact of various duplication levels on model performance. Table 1 shows the results of perplexity (PP) and improvement rate (IR). IR refers to the percentage of improvement in perplexity after deduplication. For example, the Haiku dataset with GPT-2 Medium and 30% duplication shows an IR of 5.36% from 3.73 to 3.53 after deduplication. The deduplicated client datasets represent the baseline perplexity.

The blue text in the table highlights the largest improvement in perplexity, which is for the Plays dataset with GPT-2 Medium. We observe an improvement rate of 19.61% from 34.88 to 28.04 for 30% duplication. The Plays dataset in general has a higher perplexity than other datasets as it is a relatively challenging task with longer sequences and fewer training samples. The impact of duplicates in such scenarios is aggravated. We observe a similar trend with the Sonnets dataset with a 13.64% improvement rate, which is also a difficult task with few samples and relatively long sequences.

The purple text highlights the least improvement in perplexity for the Rotten Tomatoes dataset with GPT-2 Medium. The 10% duplication results in a 1.7% improvement rate from 3.6 to 3.53. Similarly, the Haiku and Short Jokes datasets exhibit smaller improvement rates across different levels of duplication, while maintaining low perplexity. We observe these trends because these datasets are relatively easier to learn with a greater number of training samples. The impact of duplicates in such cases is not as pronounced.

The red text highlights two scenarios where deduplication results in insignificant changes ($< 1\%$) to perplexity. The Plays dataset with GPT-2 Medium shows an improvement of 0.28% after deduplication with 10% duplication while the Sonnets dataset with GPT-2 Large worsens by 0.24% after deduplication with 10% duplication. The reason for the “noisy” results for the Plays and Sonnets dataset is that both datasets have very few training data samples. A 10% duplication percentage merely adds $3-5$ duplicated data points to each client. Both datasets, however, show significant improvements for 20% and 30% duplication.

Table 1: Test set perplexity (PP) and improvement rate (IR) of perplexity after deduplication.

Model	**Dataset**	**Duplication Percentage**						**Deduplicated**
		30%		20%		10%		
		PP	IR (%)	PP	IR (%)	PP	IR (%)	PP
GPT-2 Medium	Haiku	3.73	5.36	3.69	4.3	3.6	1.94	3.53
	Rotten Tomatoes	2.4	3.75	2.36	2.18	2.35	1.7	2.31
	Short Jokes	3.95	5.31	3.89	3.85	3.83	2.34	3.74
	Poetry	5.46	8.42	5.59	10.55	5.33	6.19	5.00
	IMDB	12.81	4.57	12.71	3.75	12.5	2.0	12.25
	Sonnets	15.83	13.64	15.63	12.5	14.22	4.02	13.67
	Plays	34.31	18.27	34.88	19.61	28.12	–	28.04
GPT-2 Large	Haiku	3.27	8.86	3.26	8.58	3.23	7.73	2.98
	Rotten Tomatoes	2.65	16.98	2.6	15.38	2.53	13.00	2.2
	Short Jokes	4.11	7.78	4.02	5.72	3.95	4.05	3.79
	Sonnets	8.51	5.53	8.4	4.28	8.02	–	8.04

FL Model Training Efficiency. We analyze the model training efficiency by computing the total GPU running time of all clients during training. The CPU-bound EP-MPD costs are negligible for 10 clients and at most 5,000 elements (the size of the largest dataset is 50,000 elements distributed among all clients).

Table 2 highlights the results for GPU running time. The Time column is the total GPU running time and IR is the percentage improvement compared to the running time within the deduplicated setting.

The blue text shows the largest improvement in running time of 27.95% from 33.13 minutes to 23.87 minutes in the GPT-2 Medium experiment with the Sonnets dataset. The purple text highlights the smallest improvement in running time of 7.33% from 11.87 to 11.0 minutes in the Sonnets experiment with GPT-2 Large. We observe a relatively consistent improvement trend of approximately 22%, 16%, and 8% for 30%, 20%, and 10% duplication levels respectively. The GPT-2 Medium experiments show higher running times as these models were trained longer. We observe that the running time efficiency consistently improves across all duplication levels and experiments.

Tables 1 and 2 demonstrate the effectiveness of applying EP-MPD to FL of language models. The experiments show improvements of up to 18% in perplexity and 27% in GPU running time. Deduplication significantly reduces resource usage while enhancing the quality of FL models. The RTX A6000 GPU used in our experiments consumes 0.3 kWh of energy. As FL scales to larger datasets and more clients, the energy savings with EP-MPD become even more significant, contributing to more sustainable and cost-effective FL implementations, benefiting both the environment and the end-users.

Table 2: Total GPU training time (minutes) of all clients and improvement rate (IR) of time after deduplication.

Model	**Dataset**	**Duplication Percentage**						**Deduplicated**
		30%		20%		10%		
		Time	IR (%)	Time	IR (%)	Time	IR (%)	Time
GPT-2 Medium	Haiku	111.92	22.96	105.03	17.91	95.62	9.83	86.22
	Rotten Tomatoes	162.79	21.7	151.54	15.89	138.76	8.14	127.46
	Short Jokes	396.62	27.85	338.69	15.51	313.35	8.68	286.15
	Poetry	114.28	22.65	105.48	16.2	96.85	8.74	88.39
	IMDB	2133.36	22.56	2006.94	17.68	1788.11	7.61	1652.04
	Sonnets	33.13	27.95	28.53	16.33	26.14	8.68	23.87
	Plays	31.48	22.9	29.38	17.39	26.95	9.94	24.27
GPT-2 Large	Haiku	20.89	22.98	19.28	16.55	17.7	9.1	16.09
	Rotten Tomatoes	70.74	23.08	65.26	16.63	59.91	9.18	54.41
	Short Jokes	340.75	22.93	313.65	16.27	288.86	9.08	262.63
	Sonnets	13.91	20.92	12.89	14.66	11.87	7.33	11.0

7 Asymptotic Costs Analysis

In this section, we analyze the asymptotic costs of our schemes. Table 3 summarizes the result.

7.1 Costs of EG-PSI$^{(\mathrm{I})}$ and EG-PSI$^{(\mathrm{II})}$

Computation Cost. Initially, we focus on the computation cost of parties in EG-PSI$^{(\mathrm{I})}$. In step 1b, each client $\mathcal{C}_{j,i}$ encrypts each element of its set using the symmetric key encryption `PRP`, under each key it agrees with the clients in the other group. Specifically, it invokes `PRP` $m \cdot |S_{j,i}|$ times, where m is the total number of clients in the other group. In step 3, each client $\mathcal{C}_{j,i}$ invokes `PRP` $m \cdot |R_{j,i}|$ times, where $R_{j,i}$ is the (encrypted) intersection set. Thus, the computation complexity of each $\mathcal{C}_{j,i}$ is $O(m \cdot |S_{j,i}|)$. The computation complexity of $\mathcal{TEE}$ in step 2 is $O(m \cdot |S|)$, where $|S|$ represents the maximum cardinality of sets. The only computation that $\mathcal{TEE}$ performs is searching to find duplicated values.

Table 3: Complexities of our solutions; namely, EG-PSI$^{(\text{I})}$, EG-PSI$^{(\text{II})}$, and EP-MPD. In the table, we have considered the maximum communication complexity of a client in EP-MPD (without considering the cases where the updates reduce the clients' set size). In the table, m is the number of clients in each group, $|S|$ is the maximum cardinality of sets, and $|R|$ is the maximum cardinality of sets intersection that a client has with clients of the other group.

Protocols	**Computation Cost**		**Communication Cost**	
	Client	$\mathcal{TEE}$	Client	$\mathcal{TEE}$
EG-PSI$^{(\text{I})}$	$O(m \cdot \lvert S_{j,i}\rvert)$	$O(m \cdot \lvert S\rvert)$	$O(m \cdot \lvert S_{j,i}\rvert)$	$O(m \cdot \lvert R\rvert)$
EG-PSI$^{(\text{II})}$	$O(m \cdot \lvert S\rvert)$	$O(m \cdot \lvert S\rvert)$	$O(m \cdot \lvert S_{j,i}\rvert)$	$O(m \cdot \lvert S\rvert)$
EP-MPD	$O(m \cdot \log_2(m) \cdot \lvert S\rvert)$	$O(m \cdot \log_2(m) \cdot \lvert S\rvert)$	$O(m \cdot \log_2(m) \cdot \lvert S_{j,i}\rvert)$	$O(m \cdot \log_2(m) \cdot \lvert S\rvert)$

We proceed to analyze the computation cost of parties in EG-PSI$^{(\text{II})}$. In step 2, each client $\mathcal{C}_{j,i}$ invokes $|S_{j,i}|$ times OPRF to encrypt its set elements. In steps 3 and 4, each $\mathcal{C}_{j,i}$ performs a search on $m+1$ encrypted sets, which imposes negligible costs compared to the executions of OPRF. Thus, the complexity of each $\mathcal{C}_{j,i}$ is $O(m \cdot |S|)$, where $|S|$ is the maximum cardinality of sets. $\mathcal{TEE}$ invokes an instance of OPRF in step 2 for every set of elements of each client. Therefore, its computation complexity is $O(m \cdot |S|)$, where $|S|$ is the maximum cardinality of sets.

Communication Cost. We first concentrate on the communication cost of parties in EG-PSI$^{(\text{I})}$. Each client $\mathcal{C}_{0,i}$ in step 1a transmits m keys to the clients of the other group. Thus, its communication complexity in this step is $O(m)$. However, the clients in $\mathcal{G}_1$ do not transmit any message in this step. In step 1b, each client $\mathcal{C}_{j,i}$ of each group, transmits $m \cdot |S_{j,i}|$ messages to $\mathcal{TEE}$, where the size of each message is only about 256 bits. Thus, each client's communication complexity is $O(m \cdot |S_{j,i}|)$. On the other hand, $\mathcal{TEE}$ sends to each client $R_{j,i}$ which contains the intersection the client has with the clients in the other group. Thus, the overall communication complexity of $\mathcal{TEE}$ is $O(m \cdot |R|)$, where $|R|$ is the maximum cardinality of sets intersection that a client may have with clients of the other group.

We proceed to evaluate the communication complexity of parties in EG-PSI$^{(\text{II})}$. In step 2, each $\mathcal{C}_{j,i}$ performs OPRF evaluations on $S_{j,i}$. This procedure imposes $O(|S_{j,i}|)$ communication cost to the client. In step 3, the communication complexity of each $\mathcal{C}_{j,i}$ is $O(m \cdot |S_{j,i}|)$ because it sends its encrypted set to every client in the other group. In step 2, $\mathcal{TEE}$ performs OPRF evaluations on the elements of each $\mathcal{C}_{j,i}$ using OPRF. This procedure imposes $O(m \cdot |S|)$ communication cost to $\mathcal{TEE}$.

7.2 Costs of EP-MPD

Computation Cost. In step 3, at each level d, each client $\mathcal{C}_{j,i}$ invokes an instance of EG-PSI. When EG-PSI$^{(\text{I})}$ or EG-PSI$^{(\text{II})}$ is used, then the computation complexity of $\mathcal{C}_{j,i}$ in this step is $O(m \cdot \log_2(m) \cdot |S|)$, where $|S|$ is the maximum cardinality of the clients' sets. In step 4, $\mathcal{C}_{j,i}$ updates its set. This involves simply removing items from its local set and the associated cost is negligible compared to other costs. Thus, the overall computation complexity of $\mathcal{C}_{j,i}$ is $O(m \cdot \log_2(m) \cdot |S|)$. The computation cost is dominated by the cost of executing `PRP` and OPRF if EG-PSI$^{(\text{I})}$ and EG-PSI$^{(\text{II})}$ are used respectively. The computation complexity of $\mathcal{TEE}$ in either case (when EG-PSI$^{(\text{I})}$ or EG-PSI$^{(\text{II})}$ is used) is $O(m \cdot \log_2(m) \cdot |S|)$.

Communication Cost. In step 3, at each level d, each client $\mathcal{C}_{0,i}$ transmits its updated set to an instance of EG-PSI$^{(\text{I})}$ or EG-PSI$^{(\text{II})}$. Therefore, its overall communication complexity is $O\Big(m \cdot \big(|S_{0,i}| + \sum\limits_{u=2}^{d}(|S_{0,i}| - \sum\limits_{u'=1}^{u-1}|S_{\cap}^{(u')}|)\big)\Big)$ where $d = \log_2(m)$ and $|S_{\cap}^{(u')}|$ is the size of the intersection that $\mathcal{C}_{0,i}$ has with the clients of the other group at level u'. On the other hand, at each level d, the communication complexity of each client $\mathcal{C}_{1,i}$ is $O(m \cdot \log_2(m) \cdot |S_{1,i}|)$. Note that the majority of the clients will eventually become members of group

0. Thus, their set size eventually reduces when they (i) proceed to a higher level in the tree and (ii) have intersections with other clients' sets.

The communication complexity of $\mathcal{TEE}$, when EG-PSI$^{(\text{I})}$ is used, is $O(m \cdot \log_2(m) \cdot |R|)$, where $|R|$ is the maximum cardinality of sets intersection that a client may have with clients of the other group. The communication complexity of $\mathcal{TEE}$, when EG-PSI$^{(\text{II})}$ is used, is $O(m \cdot \log_2(m) \cdot |S|)$.

8 Related Work

A detailed deduplication method has been proposed in [31] that used existing techniques of finding duplicates such as suffix array [36] and MinHash [11]. However, there were no privacy requirements in this method as the data owner is responsible for local deduplication. As we consider the problem of deduplication in a distributive setting, thus our problem statement is fundamentally different from [31].

In [29], the problem of privacy-preserving deduplication by a centralized service was discussed – there is a server that stores multiple users' files after removing duplicates. Their method involves a second server which helps in setting up keys for each user using an OPRF. Our work is different from [29] in many ways. First, we apply OPRF for encrypting the dataset elements, which allows efficient deduplication. Second, our deduplication requirement is more granular – we want entries of users' datasets to be deduplicated. If the two datasets have the same elements irrespective of their order, then at the end of our deduplication, one of the users will receive an empty dataset and the other will receive their dataset unchanged. On the other hand, since [29] considers a file as a whole (not every set element), so two files M_1 and M_2 have the same elements but in a different order will not be removed. Third, our public key-based deduplication protocol is federated as opposed to [29] which is centralized (all data is stored in a centralized server responsible for the deduplication). Fourth, [29] is a two-server setting – one server helps in deduplication and the other stores the deduplicated files. In contrast, in our protocol apart from the users, we only have one third-party.

Another work along the lines of [29] is [33] which proposed Multi-Key Revealing Encryption (MKRE) that lets a server compress the encrypted files of multiple users by removing ciphertexts related to similar plaintexts. Their construction is based on message lock encryption, therefore, it is subject to dictionary attacks [34]. Our scheme is not vulnerable to such attacks and, as a result, provides a stronger security guarantee.

The work of [9] also has pointed out the need for data processing for ML and applied PSI-type tools. However, their problem statement is to find a mismatch in the datasets belonging to two participants, whereas we are considering finding duplicates (matching data) in sets belonging to two or more parties. Additionally, their application domain is narrow as they only considered labeled data. Their end goal is to find entries in two different sets where their features match but their corresponding labels do not match. On the other hand, our protocol supports both labeled and unlabeled data.

9 Conclusion

It was established in [31] that machine learning benefits significantly from deduplication. However, it was not evident how federated learning (FL) could benefit from deduplication without compromising data privacy. Our successful and efficient solution to this problem has paved the way for further applications. Our pioneering protocols, EG-PSI$^{(\text{I})}$ and EG-PSI$^{(\text{II})}$, can be employed in any distributed system to identify and efficiently remove duplicates. Although this paper considers elements as duplicates only if they are exact copies, there are scenarios where elements that are not exact matches but are sufficiently similar should also be treated as duplicates [31]. Our current primitives can easily be extended to address this case. By applying fuzzy hash functions, which map closely related elements to the same digest, the deduplication of near-match elements can be reduced to the problem of deduplication of exact-match elements, allowing the use of EG-PSI$^{(\text{I})}$ and EG-PSI$^{(\text{II})}$ in distributed setups. While our system's application focuses on text-based learning, it can also be extended to other areas, such as image processing and speech recognition, thereby broadening its utility and impact.

References

1. Abadi, A.: Earn while you reveal: Private set intersection that rewards participants. CoRR (2023)
2. Abadi, A., Dong, C., Murdoch, S.J., Terzis, S.: Multi-party updatable delegated private set intersection. In: FC (2022)
3. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep learning with differential privacy. CCS '16 (2016)
4. Antunes, R.S., da Costa, C.A., Küderle, A., Yari, I.A., Eskofier, B.: Federated learning for healthcare: Systematic review and architecture proposal. TIST (2022)
5. Balunović, M., Dimitrov, D.I., Jovanović, N., Vechev, M.: Lamp: Extracting text from gradients with language model priors (2022), `https://arxiv.org/abs/2202.08827`
6. Bell, J.H., Bonawitz, K.A., Gascón, A., Lepoint, T., Raykova, M.: Secure single-server aggregation with (poly)logarithmic overhead. CCS (2020)
7. Ben-Efraim, A., Nissenbaum, O., Omri, E., Paskin-Cherniavsky, A.: Psimple: Practical multiparty maliciously-secure private set intersection. In: ASIA CCS (2022)
8. Bender, E.M., Gebru, T., McMillan-Major, A., Shmitchell, S.: On the dangers of stochastic parrots: Can language models be too big? In: FAccT (2021)
9. Blass, E., Kerschbaum, F.: Private collaborative data cleaning via non-equi psi. In: IEEE SP (2023)
10. Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: CCS (2017)
11. Broder, A.Z.: On the resemblance and containment of documents. In: SEQUENCES (1997)
12. Burns, J., Moore, D., Ray, K., Speers, R., Vohaska, B.: Ec-oprf: Oblivious pseudorandom functions using elliptic curves. Cryptology ePrint Archive (2017)
13. Carlini, N., Ippolito, D., Jagielski, M., Lee, K., Tramer, F., Zhang, C.: Quantifying memorization across neural language models (2023), `https://arxiv.org/abs/2202.07646`
14. Carlini, N., Tramer, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T., Song, D., Erlingsson, U., Oprea, A., Raffel, C.: Extracting training data from large language models. In: USENIX Security (2021)
15. Cheng, K., Fan, T., Jin, Y., Liu, Y., Chen, T., Papadopoulos, D., Yang, Q.: Secureboost: A lossless federated learning framework. IEEE Intelligent Systems (2021)
16. Dasu, V.A., Sarkar, S., Mandal, K.: Prov-fl: Privacy-preserving round optimal verifiable federated learning. In: Proceedings of the 15th ACM Workshop on Artificial Intelligence and Security. p. 33–44. AISec'22, Association for Computing Machinery, New York, NY, USA (2022). `https://doi.org/10.1145/3560830.3563729`, `https://doi.org/10.1145/3560830.3563729`
17. Dörre, F., Mechler, J., Müller-Quade, J.: Practically efficient private set intersection from trusted hardware with side-channels. In: ASIACRYPT. Springer (2023)
18. Fowl, L., Geiping, J., Reich, S., Wen, Y., Czaja, W., Goldblum, M., Goldstein, T.: Decepticons: Corrupted transformers breach privacy in federated learning for language models (2023), `https://arxiv.org/abs/2201.12675`
19. Freedman, M.J., Nissim, K., Pinkas, B.: Efficient private matching and set intersection. In: EUROCRYPT (2004)
20. Ghosh, S., Nilges, T.: An algebraic approach to maliciously secure private set intersection. In: EUROCRYPT (2019)
21. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press (2004)
22. Hard, A., Rao, K., Mathews, R., Beaufays, F., Augenstein, S., Eichner, H., Kiddon, C., Ramage, D.: Federated learning for mobile keyboard prediction. CoRR (2018)
23. Ilyas, I.F., Chu, X.: Data Cleaning. Association for Computing Machinery, New York, NY, USA (2019)
24. Inbar, R., Omri, E., Pinkas, B.: Efficient scalable multiparty private set-intersection via garbled bloom filters. In: SCN (2018)
25. Jiang, J.C., Kantarci, B., Oktug, S., Soyata, T.: Federated learning in smart city sensing: Challenges and opportunities. Sensors (2020)
26. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: FC (2014)
27. Kandpal, N., Wallace, E., Raffel, C.: Deduplicating training data mitigates privacy risks in language models (2022), `https://arxiv.org/abs/2202.06539`
28. Karamolegkou, A., Li, J., Zhou, L., Søgaard, A.: Copyright violations and large language models (2023), `https://arxiv.org/abs/2310.13771`

29. Keelveedhi, S., Bellare, M., Ristenpart, T.: Dupless: Server-aided encryption for deduplicated storage. In: USENIX Security (2013)
30. Kolesnikov, V., Matania, N., Pinkas, B., Rosulek, M., Trieu, N.: Practical multi-party private set intersection from symmetric-key techniques. In: CCS (2017)
31. Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., Carlini, N.: Deduplicating training data makes language models better. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (2022)
32. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization (2019), `https://arxiv.org/abs/1711.05101`
33. Lucani, D.E., Nielsen, L., Orlandi, C., Pagnin, E., Vestergaard, R.: Secure generalized deduplication via multi-key revealing encryption. In: SCN (2020)
34. Lucani, D.E., Nielsen, L., Orlandi, C., Pagnin, E., Vestergaard, R.: Secure generalized deduplication via multi-key revealing encryption. Cryptology ePrint Archive, Paper 2020/799 (2020), `https://eprint.iacr.org/2020/799`
35. Maas, A.L., Daly, R.E., Pham, P.T., Huang, D., Ng, A.Y., Potts, C.: Learning word vectors for sentiment analysis. In: Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies (2011)
36. Manber, U., Myers, E.W.: Suffix arrays: a new method for on-line string searches. SIAM J. Comput. (1993)
37. Mattern, J., Mireshghallah, F., Jin, Z., Schölkopf, B., Sachan, M., Berg-Kirkpatrick, T.: Membership inference attacks against language models via neighbourhood comparison. In: ACL (2023)
38. McMahan, H.B., Moore, E., Ramage, D., y Arcas, B.A.: Federated learning of deep networks using model averaging. CoRR (2016)
39. Nasr, M., Carlini, N., Hayase, J., Jagielski, M., Cooper, A.F., Ippolito, D., Choquette-Choo, C.A., Wallace, E., Tramèr, F., Lee, K.: Scalable extraction of training data from (production) language models (2023)
40. Nevo, O., Trieu, N., Yanai, A.: Simple, fast malicious multiparty private set intersection. In: CCS (2021)
41. Pass, R., Shi, E., Tramèr, F.: Formal abstractions for attested execution secure processors. In: EUROCRYPT (2017)
42. Patterson, D.A., Gonzalez, J., Le, Q.V., Liang, C., Munguia, L., Rothchild, D., So, D.R., Texier, M., Dean, J.: Carbon emissions and large neural network training. CoRR (2021)
43. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I.: Language models are unsupervised multitask learners (2019)
44. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. (2020)
45. Raghuraman, S., Rindal, P.: Blazing fast PSI from improved OKVS and subfield VOLE. In: CCS (2022)
46. Rashid, M.R.U., Dasu, V.A., Gu, K., Sultana, N., Mehnaz, S.: Fltrojan: Privacy leakage attacks against federated language models through selective weight tampering (2024), `https://arxiv.org/abs/2310.16152`
47. Strubell, E., Ganesh, A., McCallum, A.: Energy and policy considerations for deep learning in NLP. In: ACL (2019)
48. Suri, A., Kanani, P., Marathe, V.J., Peterson, D.W.: Subject membership inference attacks in federated learning (2023), `https://arxiv.org/abs/2206.03317`
49. Tramèr, F., Zhang, F., Lin, H., Hubaux, J., Juels, A., Shi, E.: Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In: EuroS&P (2017)
50. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need. CoRR **abs/1706.03762** (2017), `http://arxiv.org/abs/1706.03762`
51. Xia, Q., Ye, W., Tao, Z., Wu, J., Li, Q.: A survey of federated learning for edge computing: Research problems and solutions. High-Confidence Computing (2021)
52. Xu, R., Baracaldo, N., Zhou, Y., Anwar, A., Ludwig, H.: Hybridalpha. AISec (2019)
53. Zhang, E., Liu, F., Lai, Q., Jin, G., Li, Y.: Efficient multi-party private set intersection against malicious adversaries. In: CCSW (2019)
54. Zhang, F., Zhang, H.: Sok: A study of using hardware-assisted isolated execution environments for security. In: Proceedings of the Hardware and Architectural Support for Security and Privacy 2016 (2016)

A Related Work in PSI Research

Since their introduction in [19], numerous PSIs have been designed. In general, PSIs fall into two broad categories *traditional* and *delegated*. In traditional PSIs, clients (i.e., data owners) compute the result interactively using their data stored locally. In this research line, researchers in [45] introduced two two-party

PSIs, one secure against semi-honest and the other against malicious (or active) adversaries. These protocols are the fastest two-party PSIs to date. They use Oblivious Key-Value Stores (OKVS) data structure and Vector Oblivious Linear Evaluation (VOLE). These solutions' computation cost is $O(c)$, where c is a set's cardinality. They also impose $O(c \log c^2 + \kappa)$ and $O(c \cdot \kappa)$ communication costs in the semi-honest and malicious models respectively, where κ is a security parameter.

Dorre et al. [17] proposed an efficient two-party PSI that requires each party to locally possess a secure hardware, a requirement that may not always be possible to meet. The scheme further relies on a hash function and deterministic symmetric-key encryption. The scheme's communication complexity is $O(poly(\kappa) + |S|\lambda)$ and its computation complexity is $O((2 \cdot |S|) \cdot poly(\kappa \cdot l))$, where l is a set element's bit size and λ is a security parameter. As the authors stated, the protocol cannot directly support multiple more than two parties.

Furthermore, researchers developed PSIs that allow multiple (i.e., more than two) parties to compute the intersection. The multi-party PSIs in [24,30] are secure against semi-honest adversaries while those proposed in [7,20,53,30,40] were developed to remain secure against active ones. The protocols in [30] and [40] are the most efficient multi-party PSIs designed to be secure against passive and active adversaries respectively. The computation and communication complexities of the PSI in [30] are $O(c \cdot m^2 + c \cdot m)$ and $O(c \cdot m^2)$. The PSI in [40] has a parameter t that determines how many parties can collude with each other and must be set before the protocol's execution, where $t \in [2, m)$. Its computation and communication complexities are $O(c \cdot \kappa(m + t^2 - t(m + 1)))$ and $O(c \cdot m \cdot \kappa)$ respectively.

Delegated PSIs use cloud computing for computation and/or storage. These PSIs can be further categorized into those that allow *one-off* and *repeated* delegation of PSI computation. The most efficient one that supports one-off delegation is [26], designed for the two-party setting, with an overhead of $O(c)$. Researchers also proposed a multi-party PSI that supports repeated delegation and efficient *updates* [2]. It imposes $O(h \cdot d^2 \cdot m)$ and $O(h \cdot d \cdot m)$ computation and communication costs respectively, during the PSI computation. Furthermore, to incentivize clients to participate in a PSI protocol and share their sensitive inputs, researchers proposed a PSI that rewards participants [1].

Thus, despite the development of numerous two-party and multiple-party PSIs, none of them to date can match the features offered by our PSIs.

B Datasets and FL Hyperparameters

We use the following datasets in our study:

1. *Haiku*[5]: This dataset contains 15,281 Haikus scraped from "reddit.com/r/haiku". Haiku is a Japanese style of short poetry.
2. *Rotten Tomatoes*[6]: It contains 10,662 movie reviews from the Rotten Tomatoes movie critic website. Originally intended for sentiment analysis, the dataset has 5,331 positive and negative reviews each. In our experiments, we ignore the sentiment of the reviews and train the CLM only on the review.
3. *Short Jokes*[7]: It contains 231,657 jokes scraped from Reddit. We randomly sample 50,000 jokes for our experiments.
4. *Poetry*[8]: It contains 573 poems from the Renaissance and Modern eras by various poets.
5. *IMDB* [35]: This dataset contains 50,000 movie reviews from the IMDB movie critic website. Similar to the Rotten Tomatoes dataset, the IMDB dataset was originally intended for sentiment analysis. We ignore the sentiment of each review and only train the CLM on the review.
6. *Sonnets*[9]: This dataset contains 460 sonnets (including variations) from William Shakespeare. Sonnets are 14-line poems with variable rhyme schemes

[5] `https://www.kaggle.com/datasets/bfbarry/haiku-dataset`

[6] `https://huggingface.co/datasets/cornell-movie-review-data/rotten_tomatoes`

[7] `https://www.kaggle.com/datasets/abhinavmoudgil95/short-jokes`

[8] `https://huggingface.co/datasets/merve/poetry`

[9] `https://huggingface.co/datasets/Lambent/shakespeare_sonnets_diffused`

7. *Plays*[10]: It contains 521 plays from William Shakespeare.

We train the GPT-2 Medium and GPT-2 Large models for 5–10 and 1–2 local epochs respectively. Both models are trained for 5 rounds. The Haiku GPT-2 Large was trained for 1 epoch while the other GPT-2 Large models were trained for 2 epochs. The Poetry and Sonnets GPT-2 Medium models were trained for 10 epochs while the others were trained for 5 epochs. We observe that training GPT-2 Large for more epochs and data results in overfitting and poor Test set perplexity across all duplication levels.

For all experiments, we use a learning of 5×10^{-5} with the AdamW [32] optimizer. We use a linear warm-up schedule with 10% of the training steps as warm-up steps. We use ℓ_2-norm clipping for the gradients and set the threshold to 1.0. We set the sequence length for the datasets as follows:

1. Haiku: 100
2. Rotten Tomatoes: 200
3. Short Jokes: 100
4. Poetry: 1000
5. IMDB: 500
6. Sonnets: 400
7. Plays: 1000

For the *Short Jokes* dataset, we randomly sample 50,000 elements to ensure the running time is feasible as we train with 10 clients. We set the batch size to values in the range $[4, 32]$, depending on the sequence length and model size to ensure that training can be performed on one RTX A6000 GPU. We use a Train/Test split of 80/20 for all datasets. In the case of datasets like IMDB that have a separate Test set, we combine the original Train and Test sets into a single dataset, shuffle it, and then create 80:20 splits. We fix the random seed to 123 for the Numpy, PyTorch, and Python random number generators to ensure the reproducibility of our results.

[10] `https://huggingface.co/datasets/Trelis/tiny-shakespeare`