Fuzzy Private Set Intersection with Large Databases via Locality-Sensitive Hashing*

Francesca Falzon
Lucerne University of Applied Sciences and Arts
Lucerne, Switzerland

Andreas Athanasiou
TU Delft
Delft, The Netherlands

Laura Hetz
ETH Zurich
Zurich, Switzerland

Tjitske Ollie Koster
TU Delft
Delft, The Netherlands

Evangelia Anna Markatou
TU Delft
Delft, The Netherlands

ABSTRACT

Many query workloads require privacy-preserving similarity joins: parties, each holding a private database, may want to learn which of their records are close to records in the other party's database under a distance metric, without leaking anything beyond this information. Fuzzy private set intersection (fuzzy PSI) is a cryptographic primitive that addresses this problem. In this setting, a client and a server hold sets X and Y, respectively, and the client learns the subset of points in X that are "close" to points in Y. Fuzzy PSI has many use cases, including biometric authentication, password breach monitoring, contact tracing, regional-outbreak monitoring, and machine-learning label matching. Despite growing interest in fuzzy PSI, many existing schemes still require communication costs and client computation linear in the server's set size $|Y|$. Consequently, these protocols do not scale well to the unbalanced setting, where $|X| \ll |Y|$, as is common in many client-to-server applications.

In this work, we propose an efficient fuzzy PSI protocol for the unbalanced setting. At its core is a fuzzy probabilistic data structure (fuzzy PDS), which generalizes prior work by combining locality-sensitive hashing (LSH) with compact filters, enabling tunable false-positive and false-negative rates while ensuring efficient queries. Building on techniques from unbalanced PSI, we combine oblivious pseudorandom functions and private information retrieval with our fuzzy PDS to support private and communication-efficient approximate matching. We instantiate the protocol with an LSH for the L_2 metric and evaluate it on two real-world workloads: regional-outbreak monitoring and machine-learning label matching. Our results show significant reductions in communication and runtime compared with state-of-the-art fuzzy PSI schemes, with gains that increase as the server database size $|Y|$ grows.

1 INTRODUCTION

As sensitive data increasingly moves to cloud services and cross-organizational platforms, there is a growing need to support privacy-preserving database queries. One simple, yet important example is an equi-join across two privately held datasets, in which the parties should learn only the matching records and nothing else. *Private set intersection (PSI)* is a widely studied cryptographic primitive that addresses this need over single-key records: it allows two parties, holding input sets X and Y, to compute $X \cap Y$ without revealing any additional information [48, 66]. *Unbalanced PSI* is a special case of PSI that often arises in practice, in which a lightweight client $\mathbf{C}$ holds a much smaller set X than that of the server's database, i.e., $|X| \ll |Y|$. Unbalanced PSI has numerous applications, including mobile private contact discovery [29, 33, 35], password breach monitoring [38], and private pattern matching and contact tracing in medical contexts [3, 11, 17, 55, 56].

Exact equality, however, is often too restrictive. In many real-world use cases, a relaxed notion is needed, where records match if they are close under an application-specific metric. For example, during regional outbreaks of mosquito-borne diseases, an individual may wish to privately determine whether their location is approximately close to a reported case, without requiring an exact coordinate match. Similarly, medical or biometric records, such as retinal scans of the same eye, may be close even when their feature vectors are not identical. In such settings, exact PSI can miss relevant matches. The same issue arises in privacy-preserving record linkage, where parties seek to match records corresponding to the same entity across databases despite inconsistencies or errors in attributes (e.g., a misspelled name) [60, 62].

Fuzzy PSI (e.g., [10, 14, 21, 22, 49, 57, 58]) addresses this problem by enabling parties to privately identify client elements that are "close" to some element in the server's set under an appropriate, application-specific notion of closeness. More formally, given a metric space with distance function $\mathsf{dist}(\cdot,\cdot)$, a client set X, and a server set Y, the goal of fuzzy PSI is for the client to learn the subset of values $x \in X$ for which there exists $y \in Y$ such that $\mathsf{dist}(x, y)$ is small, while leaking no additional information about either party's values that are not in the intersection. Fuzzy PSI has received a lot of attention in recent years and, in order to avoid the naive approach of privately testing the fuzzy matching predicate for every pair in $X \times Y$, various techniques have been proposed including fuzzy matching based on oblivious pseudorandom functions [59], distance-aware random OTs [49, 51], incremental function secret sharing [22], and prefix encodings of set elements [10, 14].

Another natural approach is to use *locality-sensitive hashing (LSH)* [32]. LSH functions are commonly used in approximate matching protocols with large databases [25, 31, 43, 61, 62]. An LSH function hashes nearby points to the same value with high probability, and hashes far points to the same value with low probability. LSH is used for problems such as private record linkage [62] and approximate nearest-neighbor search [67], both in privacy-preserving settings [52, 62] and in plaintext database systems where scalability is paramount [25, 31, 43, 61]. LSH has also been used in the context

*This is the full version of work to appear at VLDB 2027.

of fuzzy PSI: explicitly, as a preprocessing step to coarsely filter candidates before applying a more expensive fuzzy matching procedure [58]; and implicitly, through subsampling techniques similar to random-sampling LSH [57].

Existing techniques focus on recovering the exact fuzzy intersection (i.e., with negligible errors), but often do not scale well to large databases. This motivates the following question:

> *Can we design an unbalanced fuzzy PSI protocol for billion-element server datasets, trading efficiency at the cost of a small tunable error rate?*

1.1 Contributions

We answer this question in the affirmative. We propose an efficient fuzzy PSI protocol for the unbalanced setting. The server performs preprocessing over its large set once, and this cost can be amortized across many clients. Our starting point is the observation that state-of-the-art unbalanced PSI protocols [29, 42] combine compact filter-based data structures with PIR: the server preprocesses its large set once, and the client privately retrieves only the entries relevant to its much smaller query set.

We extend this paradigm from exact to fuzzy matching by incorporating an LSH-based filter abstraction that supports private approximate-membership queries. This preserves the scalability of filter-based unbalanced PSI while incurring only a small, tunable error probability from the LSH and probabilistic data structures.

Our contributions can be summarized as follows.

(C1) A data structure for fuzzy membership queries (§ 3). We formalize *fuzzy probabilistic data structures* (fuzzy PDSs) as a generalization of prior work on probabilistic data structures for approximate membership queries e.g., [25, 34]. Our framework combines LSH with compact filter-based data structures such as cuckoo filters (CFs). At a high level, nearby points collide with high probability under the LSH, while the underlying filter supports efficient membership testing.

Our fuzzy PDS has a modular design that can be instantiated with any LSH family and any PDS that supports insertions and membership testing, making it adaptable to different use cases and different metrics. It also has tunable false positive probability (FPP) and false negative probability (FNP), controlled by the LSH and filter parameters. We formalize correctness and bound the FPP and FNP of our generic fuzzy PDS construction. Our construction may also be of independent interest in other applications such as private keyword search [9] and database systems [1].

(C2) A new fuzzy unbalanced PSI protocol (§4). We propose and formalize a new approach for constructing *fuzzy unbalanced PSI* protocols using a fuzzy PDS. To the best of our knowledge, this is the first approach to combine LSH-based approximate matching with filter-based unbalanced PSI techniques (e.g., [29, 33, 42]).

The server encodes its LSH values in a PRF-wrapped fuzzy PDS, while the client uses OPRF and PIR subprotocols to privately derive query tokens and retrieve matching entries. This design avoids the quadratic cost of comparing all pairs in $X \times Y$, and achieves substantially better scalability and latency than prior fuzzy PSI approaches, at the cost of tunable FPP and FNP. Our approach is also highly modular: it demonstrates how a fuzzy PDS can be integrated into existing filter-based approaches. Furthermore, while our protocol builds on prior work, we note that those works did not provide formal security proofs. We fill this gap by supporting our construction with formal proofs of correctness and security.

(C3) Implementation and evaluation (§5). We instantiate our filter-based fuzzy PSI protocol using a fuzzy PDS built from CFs and evaluate its performance across two Euclidean use cases. First, we consider low-dimensional, large-scale real-world location data for location matching in regional-outbreak monitoring. Second, we evaluate high-dimensional ML label matching. In both settings, our approach achieves reasonable performance while maintaining low error rates. We show that our protocol scales to datasets of over 1 billion elements with <3% FPP and <1% FNP in only 12.4 minutes, requiring just 0.7 GB of communication. Benchmarking against prior fuzzy PSI approaches [14, 49, 58] shows that our protocol supports significantly larger datasets and substantially outperforms existing approaches on large databases. Finally, we evaluate how our approach can be extended to data that are not naturally represented in the Euclidean space by using a feature representation.

1.2 Prior Work

Unbalanced PSI. Schemes for unbalanced PSI typically comprise (i) an *offline* preprocessing phase—independent of the client's input—in which the server preprocesses Y, and (ii) an *online* querying phase. Some systems additionally include a setup phase that is independent of all inputs [35]. To achieve communication cost sublinear in the size of the larger set Y, state-of-the-art schemes [29, 42] combine PSI and PIR.

Fuzzy Private Set Intersection. Fuzzy PSI was first proposed by Freedman et al. [20]. Chmielewski and Hoepman [13] later showed that this construction is insecure and proposed two new constructions for Hamming distance. Ye et al. [65] further improved the communication cost. Uzun et al. [57] proposed a fuzzy labeled PSI protocol for biometric inputs by modeling proximity as a t-out-of-T exact set-item match. Chakraborti et al. [10] introduced distance-aware PSI for metric-space inputs, with constructions for (i) the L_1 norm over integers using prefix encoding to succinctly represent all elements within threshold-distance, and (ii) Hamming distance over binary strings using subsampling to make communication independent of the string length. Van Baarsen and Pu [58] proposed the first black-box fuzzy PSI construction for handling polynomially large radius and dimension; one of their constructions uses LSH to bucket nearby points and then filters candidate matches using a fuzzy matching protocol for L_p distance. In follow-up work [59], the authors proposed fuzzy PSI for general Minkowski distance using cheaper operations during the online phase. Gao et al. [21] formalized fuzzy mapping (Fmap), gave instantiations for Hamming and L_∞, and described a generic Fmap-to-FPSI construction. Dang et al. [14] adapted the prefix-encoding technique of [10] for L_p and L_∞ to obtain fuzzy PSI protocols whose communication and computation scale logarithmically in the universe size n and linearly in the threshold δ and the input set sizes. Garimella et al. [23] introduced structure-aware PSI (sa-PSI) as a generalization of fuzzy PSI and proposed a semi-honest sa-PSI protocol under the L_∞ norm in d-dimensional space using Boolean function secret sharing; this was extended to a more concretely efficient sa-PSI framework [7] and to the malicious setting [24]. Rausch et al. [51] proposed a

fuzzy PSI scheme based on secret-sharing and OT for Hamming distance without input assumptions. Their scheme is concretely efficient compared to related work. Yang et al. [64] proposed two fuzzy PSI protocols for the L_∞ and L_1 metrics with communication and computation complexity linear in n, m, d, and ρ; their protocols leverage an oblivious programmable PRF with secret-shared outputs.

In parallel work, Meng et al. [46] take a similar approach to ours: they use spatial hashing with deterministic coverage guarantees for L_∞ distance to generate candidate pairs for a downstream fuzzy-matching protocol, and leverage an oblivious key-value store to efficiently represent and retrieve these candidates. In contrast, we propose a fuzzy PDS framework that generalizes probabilistic membership structures to encode approximate similarity via LSH, with tunable false-positive and false-negative rates.

Nearest Neighbor. Privacy-preserving Approximate Nearest Neighbor Search (ANNS) lets a client privately query a database for its nearest neighbor. Many techniques used in Fuzzy PSI (e.g., LSH [32]) were developed in the context of plaintext ANNS. However, existing private ANNS protocols are often unsuitable for fuzzy PSI due to their security assumptions or efficiency: [12] relies on distributed ORAM and garbled circuits, [52] requires two non-colluding servers, and [27, 70] provide only client privacy. Although Panther [41] provides both client and server privacy, its top-k output does not directly support strict threshold membership and requires additional private verification.

LSH-based data structures. Our fuzzy PDS abstraction is closest to two data structures. The LSH-based approximate nearest-neighbor structure of Gionis et al. [25] stores a database in multiple LSH hash tables and answers queries by retrieving the corresponding buckets, followed by explicit distance checks over the candidates. A variation using LSH with increasingly large radius thresholds was introduced by [52] to enable comparison-free ANN in the private setting. More broadly, LSH has been extensively studied for ANN; for example [67] combines LSH with a PM-tree index to improve candidate filtering and distance estimation.

Kirsch and Mitzenmacher [34] introduced distance-sensitive Bloom filters (BFs), which replace standard Bloom-filter hashing with LSH to answer whether a query is close to a stored element. Their analysis focuses mainly on the Hamming metric and on the distribution of matching hash locations. Hua et al. [30] similarly introduce locality-sensitive Bloom filters (LSBFs) for approximate membership queries by replacing Bloom-filter hashing with LSH and using a smaller auxiliary Bloom filter to reduce false positives. These approaches, however, introduce false negatives due to the probabilistic guarantees of LSH. Goswami et al. [26] gave upper and lower space bounds for distance-sensitive filters without false negatives and proposed theoretical constructions for Hamming-space approximate membership. Kumar et al. [39] consider fuzzy Bloom filters for approximate string comparison based on edit distance.

Our fuzzy PDS abstraction generalizes this line of work by separating the approximate hashing mechanism from the underlying probabilistic data structure, thus allowing other compact filters beyond Bloom filters to be used. This abstraction also yields tunable false-positive and false-negative bounds that apply across different choices of LSH and underlying PDS.

Wei and Kerschbaum [62] studied privacy-preserving record linkage by combining LSH with frequency smoothing and oblivious bin joins. In a non-privacy setting, [5] proposed a data compression approach whereby they replaced the hash functions in a Bloom filter with LSH. While our construction is different in its design, it also benefits from compression due to LSH output "collisions". Similar to our work, [62] trades a small error for performance gains, however, its scalability to very large databases remains unclear.

2 PRELIMINARIES

We denote by $[n]$ the set of integers $\{1, 2, \ldots, n\}$. We denote d-dimensional vectors as $\mathbf{v} = (v_1, \ldots, v_d)$.

Definition 1. *A **metric space** is an ordered pair $(\mathcal{U}, \mathrm{dist})$ where $\mathcal{U}$ is a set and $\mathrm{dist} : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ is a metric (or distance function) on $\mathcal{U}$ such that for all points $x, y, z \in \mathcal{U}$ the following axioms are satisfied: (1) Non-negativity: $\mathrm{dist}(x, y) \geq 0$, (2) Symmetry: $\mathrm{dist}(x, y) = \mathrm{dist}(y, x)$, (3) the triangle inequality: $\mathrm{dist}(x, z) \leq \mathrm{dist}(x, y) + \mathrm{dist}(y, z)$, and (4) $\mathrm{dist}(x, y) = 0$ if and only if $x = y$.*

We evaluate our protocol on the Euclidean space $(\mathbb{R}^2, \|\cdot\|_2)$ with $\|\mathbf{x} - \mathbf{y}\|_2 = \left(\sum_{i=1}^{2} |x_i - y_i|^2\right)^{1/2}$.

2.1 Locality-Sensitive Hashing

Locality-sensitive hash functions (LSHs) map high-dimensional data to a lower-dimensional space in a way that preserves approximate similarity. This is extremely useful in large databases [25, 31, 43, 61, 62]. Collisions structure the data so that nearby points collide with high probability, while far-apart points collide with low probability. We formalize this below:

Definition 2 ([32]). *A family of functions $\mathcal{H} = \{H : \mathcal{U} \to \mathcal{V}\}$ is said to be a (r_i, r_o, p_1, p_2)-**sensitive locality-sensitive hash (LSH)** family with respect to metric space $(\mathcal{U}, \mathrm{dist})$ if $r_i < r_o$, $p_1 > p_2$ and for any $x, y \in \mathcal{U}$*

- *if $\mathrm{dist}(x, y) \leq r_i$ then $\Pr_{\mathcal{H}}[H(x) = H(y)] \geq p_1$, and*
- *if $\mathrm{dist}(x, y) \geq r_o$ then $\Pr_{\mathcal{H}}[H(x) = H(y)] \leq p_2$.*

In this definition, r_i is the inner radius (nearby) and r_o the outer radius (far-apart). Note that no guarantees are given for points that lie between the inner and outer radii. As our constructions rely on LSH, we similarly cannot provide guarantees for fuzzy matches at distances between the inner and outer radii.

LSH constructions often use two additional parameters, which we denote by K and L. At a high level, we sample $K \cdot L$ hash functions $\{H_{\ell,k}\}_{\ell \in [L], k \in [K]}$ from the underlying LSH family. For each $\ell \in [L]$, we define the concatenated hash value $G_\ell(\mathbf{x}) = \big(H_{\ell,1}(\mathbf{x}), \ldots, H_{\ell,K}(\mathbf{x})\big)$. We say that two points $\mathbf{x}$ and $\mathbf{y}$ collide if there exists some $\ell \in [L]$ such that $G_\ell(\mathbf{x}) = G_\ell(\mathbf{y})$. We refer to this as an **AND/OR construction**. All K hash functions should give the same output (AND), for at least one $\ell^* \in [L]$ (OR). This construction allows flexibility in optimizing the FP/FN rates.

2.2 Oblivious PRFs

A batch oblivious pseudorandom function (OPRF) is a two-party protocol in which the server holds a PRF key $\mathsf{k}_{\mathsf{PRF}}$ for the function $F_{\mathsf{k}_{\mathsf{PRF}}}$, and the client holds a batch of inputs $\mathbf{x} = (x_1, \ldots, x_{n_{\mathsf{bSize}}})$.

Parameters: Batch size n_{bSize}.
Input: The client **C** holds a vector $\mathbf{x} = (x_1, \ldots, x_{n_{\mathsf{bSize}}}) \in \mathcal{U}^{n_{\mathsf{bSize}}}$, and the server **S** holds a key $\mathsf{k}_{\mathsf{PRF}} \leftarrow \{0,1\}^\lambda$ for a PRF

$$F : \mathcal{K} \times \mathcal{U} \to \mathbb{F}.$$

Functionality:

1. Receive the vector $\mathbf{x}$ from client **C**.
2. Receive PRF key $\mathsf{k}_{\mathsf{PRF}}$ from server **S**.
3. For each $i \in [n_{\mathsf{bSize}}]$, compute $v_i := F(\mathsf{k}_{\mathsf{PRF}}, x_i)$.
4. Output $\mathbf{v} = (v_1, \ldots, v_{n_{\mathsf{bSize}}})$ to **C** and n_{bSize} to **S**.

Figure 1: Ideal functionality for a batch OPRF, $\mathcal{F}_{\mathsf{OPRF}}$.

Without learning $\mathsf{k}_{\mathsf{PRF}}$, the client obtains the PRF evaluations

$$\mathbf{v} = (F_{\mathsf{k}_{\mathsf{PRF}}}(x_1), \ldots, F_{\mathsf{k}_{\mathsf{PRF}}}(x_{n_{\mathsf{bSize}}})),$$

while the server learns only the public batch size n_{bSize}.

In addition to privacy, we also require that the OPRF satisfies *pseudorandomness*. We omit the formal definition, but intuitively, this means that the values learned by any efficient client from Π_{OPRF} are computationally indistinguishable from those obtained by querying a truly random function on the same inputs. For a comprehensive overview of OPRFs, see [8].

Definition 3. *Let* $\mathsf{View}^{\Pi}_{\mathbf{C}}(X, \mathsf{k}_{\mathsf{PRF}})$ *and* $\mathsf{View}^{\Pi}_{\mathbf{S}}(X, \mathsf{k}_{\mathsf{PRF}})$ *denote the view of the client* **C** *and server* **S** *in a batch OPRF protocol* Π*, respectively. Let* $\mathsf{Out}^{\Pi}(X, \mathsf{k}_{\mathsf{PRF}})$ *be the output of* **C** *in the protocol. Let* $O = f(X, \mathsf{k}_{\mathsf{PRF}}) = \{F(\mathsf{k}_{\mathsf{PRF}}, x_i)\}_{i \in [n_{\mathsf{bSize}}]}$ *denote the output of* **C** *in the ideal functionality* $\mathcal{F}_{\mathsf{OPRF}}$ *(Fig. 1).*

A batch OPRF Π ***securely realizes*** *the functionality* $\mathcal{F}_{\mathsf{OPRF}}$ *in the presence of* ***semi-honest adversaries*** *if there exist probabilistic polynomial-time (PPT) simulators* $\mathsf{Sim}_{\mathbf{C}}$ *and* $\mathsf{Sim}_{\mathbf{S}}$ *such that for all inputs* X *and* $\mathsf{k}_{\mathsf{PRF}}$*, the following distributions are computationally indistinguishable:*

$$\mathsf{View}^{\Pi}_{\mathbf{C}}(\mathbf{x}, \mathsf{k}_{\mathsf{PRF}}) \overset{c}{\approx} \mathsf{Sim}_{\mathbf{C}}(1^\lambda, \mathbf{x}, O)$$

$$(\mathsf{View}^{\Pi}_{\mathbf{S}}(\mathbf{x}, \mathsf{k}_{\mathsf{PRF}}), \mathsf{Out}^{\Pi}(\mathbf{x}, \mathsf{k}_{\mathsf{PRF}})) \overset{c}{\approx} (\mathsf{Sim}_{\mathbf{S}}(1^\lambda, n_{\mathsf{bSize}}, \mathsf{k}_{\mathsf{PRF}}), O).$$

2.3 Probabilistic Data Structures

A probabilistic data structure (PDS) is a compact data structure that supports answering different query types with probabilistic guarantees. In this work we focus on PDS for approximate membership queries (AMQ). We refer to such structures as AMQ-PDSs, for example Bloom filters [6] and cuckoo filters [18].

In this work, we restrict ourselves to insertion-only PDSs, i.e., elements can be inserted but not deleted. We define the syntax of an insertion-only pseudorandom function (PRF)-wrapped PDS in Def. 4 based on [19]. The query functionality of [19] is split into two algorithms, GetInfo and CheckMemb, to allow the generic use of this syntax in the context of privacy-preserving applications.

Definition 4. *Let* Σ *be an* ***insertion-only PRF-wrapped probabilistic data structure (PDS)****, let* sp *be its setup parameters, and let* $F_{\mathsf{k}_{\mathsf{PRF}}}$ *be a keyed PRF. The public parameters are implicit inputs to all algorithms.* Σ *comprises five algorithms with the following syntax:*

- $(\mathsf{pp}, D) \leftarrow \Sigma.\mathsf{Setup}(\mathsf{sp}, F_{\mathsf{k}_{\mathsf{PRF}}}, m)$*: takes setup parameters* sp*, keyed PRF* $F_{\mathsf{k}_{\mathsf{PRF}}}$ *and input set size* m*; it outputs a data structure* D *and public parameters* pp*.*
- $(\mathsf{locs}, \mathsf{info}) \leftarrow \Sigma.\mathsf{GetInfo}(F_{\mathsf{k}_{\mathsf{PRF}}}(x))$*: takes as input the PRF value* $y = F_{\mathsf{k}_{\mathsf{PRF}}}(x)$*; it outputs location indices* locs *and information* info*.*
- $D' \leftarrow \Sigma.\mathsf{Insert}(D, \mathsf{locs}, \mathsf{info})$*: takes as input the data structure* D*, location indices* locs*, and information* info*; it returns the updated data structure* D'*.*
- $\mathsf{resp} \leftarrow \Sigma.\mathsf{Query}(D, \mathsf{locs})$*: takes as input the data structure* D *and location indices* locs*; it returns values* resp*.*
- $b \leftarrow \Sigma.\mathsf{CheckMemb}(\mathsf{resp}, \mathsf{info})$*: takes values* resp *and information* info*; it returns a bit* $b \in \{0,1\}$*.*

Checking whether an element x is a member of a PDS D (i.e., querying for x) is split into two algorithms: Σ.GetInfo, which returns the information needed to test membership in the data structure, and Σ.CheckMemb, which uses this information to query the data structure and determine membership. We leave the output of GetInfo—denoted as info—abstract as this is highly implementation-specific; for example, in a cuckoo filter, info comprises bucket indices and a tag, whereas in a Bloom filter, it comprises only the indices of the relevant bits in the array.

Intuitively, correctness states that if an item was inserted into the PDS, querying for it results in a positive membership test with probability $1 - \epsilon$ where ϵ is the false positive rate (FPR) of the PDS. If an item is not in the PDS, the query returns a negative result with probability 1, i.e., there are no false negatives.

A PRF-wrapped PDS runs—as the name implies—every item through a keyed-PRF and inserts the PRF output into the PDS. Hence, the inserted items are pseudorandom, and their location and value in the data structure do not reveal any information about the original item. Privacy of the PRF-wrapped PDS reduces to PRF indistinguishability [19].

2.4 Private Information Retrieval

Schemes for *private information retrieval (PIR)* allow a client to query a public database without revealing the queried record to the server holding this database. For the server to stay oblivious, it must "touch" every database record when answering the query. This per-query computational overhead limits practical use cases; hence, recent work has focused on shifting the computation to a preprocessing phase, e.g., [28, 36, 40, 68, 69]. The server and client jointly preprocess the database to compute a hint that enables the client to efficiently query the database later. We give a definition for single-server PIR in Def. 5.

Definition 5. *A single-server private information retrieval (PIR) protocol* Π_{PIR} *for a database* $D \in \mathcal{D}^N$*, consists of the following algorithms, all taking the security parameter* λ *and public parameters* pp *as implicit inputs.*

- $\mathsf{pp} \leftarrow \Pi_{\mathsf{PIR}}.\mathsf{GenPP}(1^\lambda, N)$ *takes as input security parameter* λ *and database size* N*; it outputs public parameters* pp*.*
- $(\mathsf{st}_C; \mathsf{st}_S) \leftarrow \Pi_{\mathsf{PIR}}.\mathsf{Setup}(\perp; D)$ *an interactive protocol between a client and server. The client inputs* $\perp$ *and the server inputs the database* D*. The protocol outputs states* st_C *and* st_S *to client and server respectively.*

Parameters: Metric space $(\mathcal{U}, \mathsf{dist})$, thresholds $\delta_i < \delta_o$, probabilities $\psi_1 > \psi_2$, probabilities $\rho_1 > \rho_2$, parameter L, client set size n, and server set size m. Let MatchLeak be a $(\delta_i, \delta_o, \psi_1, \psi_2, \rho_1, \rho_2, L)$-valid match distribution.

Input: Client set $X \subseteq \mathcal{U}$ and server set $Y \subseteq \mathcal{U}$.

Functionality:

1. Receive input X from client **C**.
2. Receive input Y from server **S**.
3. Sample a map $\mathsf{match} \leftarrow\!\!\$\ \mathsf{MatchLeak}(X, Y)$.
4. Compute $Z = \{x \in X : \mathsf{match}[x] \neq 0^L\}$.
5. Output Z, match, and $|Y|$ to the client **C**.
6. Output $|X|$ to the server **S**.

Figure 2: Ideal functionality of Fuzzy PSI. Note that if $L = 1$, then the probabilities collapse to $\psi_1 = \rho_1$ and $\psi_2 = \rho_2$.

- $(\mathsf{st}'_C, \mathsf{resp}; \perp) \leftarrow \Pi_{\mathsf{PIR}}.\mathsf{Query}(\mathsf{st}_C, \mathsf{idx}; \mathsf{st}_S, D)$ *an interactive protocol between a client and server. The client inputs state* st_C *and index* $\mathsf{idx} \in [N]$*, the server inputs state* st_S *and database* D*. The protocol outputs the updated state* st'_C *and a response* $\mathsf{resp} \in \mathcal{D}$ *to the client, and* $\perp$ *to the server.*

A PIR scheme should satisfy correctness and security. See Appendix A.1 for formal definitions of these properties.

This work uses PIR as a building block and considers the state-of-the-art BatchPIR [47] scheme for evaluation, a single-server scheme with query batching and without client-preprocessing.

2.5 Fuzzy Private Set Intersection

Let $(\mathcal{U}, \mathsf{dist})$ be a metric space. Fuzzy private set intersection (Fuzzy-PSI) enables two parties to privately compute approximate matches of their sets and is parameterized by six values: a closeness threshold δ_i (inner radius), a distance threshold δ_o (outer radius), and probabilities $\rho_1 > \rho_2$ and $\psi_1 > \psi_2$. We consider a client **C** holding a set $X = \{x_1, \ldots, x_n\} \subset \mathcal{U}$ and a server **S** holding a set $Y = \{y_1, \ldots, y_m\} \subset \mathcal{U}$. Specifically, we focus on the *unbalanced setting*, in which set Y is significantly larger than set X (i.e., $|X| \ll |Y|$).

At a high level, the goal is for the client to learn which of its elements are close to the server's set. In our functionality (Fig. 2), this information is represented by a *match dictionary* $\mathsf{match} : X \to \{0,1\}^L$. For each $x \in X$, the vector $\mathsf{match}(x)$ records which of the L repetitions produced an accepting response for x. The client receives Z, match, and $|Y|$, where the fuzzy intersection is given by

$$Z = \{x \in X : \mathsf{match}(x) \neq 0^L\}.$$

Informally, if some $y \in Y$ is close to x, i.e., $\mathsf{dist}(x, y) \leq \delta_i$, then x is included in Z with probability at least ρ_1. Conversely, if x is far from every $y \in Y$, i.e., $\mathsf{dist}(x, y) \geq \delta_o$ for all $y \in Y$, then x is included in Z with probability at most ρ_2. Note that the functionality is undefined in the case where some $y \in Y$ is neither close enough to nor far enough from x, i.e., $\delta_i \leq \mathsf{dist}(x, y) \leq \delta_o$.

Before we formalize our functionality, we first introduce the notion of a valid match distribution.

Definition 6. *Let* $(\mathcal{U}, \mathsf{dist})$ *be a metric space, let* $\delta_i < \delta_o$*, let* $L \in \mathbb{N}$*, and let* $\psi_1 > \psi_2$ *and* $\rho_1 > \rho_2$ *be probabilities. A randomized algorithm* $\mathsf{MatchLeak}(X, Y)$ *is a* $(\delta_i, \delta_o, \psi_1, \psi_2, \rho_1, \rho_2, L)$***-valid match distribution*** *if on input of* $X, Y \subseteq \mathcal{U}$ *it outputs a map* $\mathsf{match} : X \to \{0,1\}^L$ *such that for every* $x \in X$*:*

- *If there exists* $y \in Y$ *such that* $\mathsf{dist}(x, y) \leq \delta_i$*, then*

$$\forall \ell \in [L],\ \Pr[\mathsf{match}(x)[\ell] = 1] \geq \psi_1,$$
$$\text{and} \quad \Pr[\mathsf{match}(x) \neq 0^L] \geq \rho_1.$$

- *If for all* $y \in Y$ *we have* $\mathsf{dist}(x, y) \geq \delta_o$*, then*

$$\forall \ell \in [L],\ \Pr[\mathsf{match}(x)[\ell] = 1] \leq \psi_2,$$
$$\text{and} \quad \Pr[\mathsf{match}(x) \neq 0^L] \leq \rho_2.$$

For points $x \in X$ for which for all $y \in Y$: $\delta_i < \mathsf{dist}(x, y)$, but there exists $y \in Y$ with $\delta_i < \mathsf{dist}(x, y) < \delta_o$, no guarantee is given.

We formalize the fuzzy PSI functionality in Fig. 2. Our definition is similar to that of [57], and is naturally compatible with LSH-based constructions. Note that we adopt a different parameter notation from the LSH definition to underscore that the protocol parameters are distinct from the underlying LSH parameters (though they may be a function of them). The client has access to MatchLeak, which reveals more information than returning the set of fuzzy matches. This leakage is a conscious trade-off: revealing this information allows us to construct substantially more efficient protocols.

We emphasize that when $L = 1$, the match leakage reduces to the standard leakage, in which an item is revealed only as either "close" or "far." Using $L > 1$ provides additional information through the match leakage, but enables more efficient constructions. Alternatively, the match leakage can be hidden using stronger, but consequently more expensive, cryptographic protocols.

We note that in our fuzzy PSI protocol (Fig. 4), the MatchLeak parameters depend on the underlying LSH; in particular, the number of matches out of L directly reflects the number of LSH matches. For many common LSH schemes (e.g., E2LSH [15] and bit-sampling LSH [32]), closer elements are more likely to produce matching outputs across hash functions from the same family. Thus, when $L > 1$, closer elements are more likely to satisfy $\mathsf{match}(x)[\ell]$ for more $\ell \in [L]$. For example, if the sets contain location data, then more matches implies that the two locations are likely closer. However, the definition of an LSH (Def. 2) only bounds collision probabilities and does not guarantee how outputs behave for closer or farther points. Thus, Def. 2 does not capture this behavior and we cannot make a general claim about all LSH schemes.

Definition 7. *Let* $\mathsf{View}^{\Pi}_{\mathbf{C}}(X, Y)$ *and* $\mathsf{View}^{\Pi}_{\mathbf{S}}(X, Y)$ *denote the views of the client* **C** *and server* **S** *in a real execution of a protocol* Π*, respectively. Let* $\mathsf{Out}^{\Pi}_{\mathbf{C}}(X, Y)$ *and* $\mathsf{Out}^{\Pi}_{\mathbf{S}}(X, Y)$ *denote the parties outputs in the real protocol execution. Let* $(O_{\mathbf{C}}, O_{\mathbf{S}}) \leftarrow \mathcal{F}_{\mathsf{FPSI}}(X, Y)$ *denote the outputs of the ideal functionality* $\mathcal{F}_{\mathsf{FPSI}}$ *in Fig. 2.*

A fuzzy private set intersection protocol Π ***securely realizes*** $\mathcal{F}_{\mathsf{FPSI}}$ *in the presence of* ***semi-honest adversaries*** *if there exist PPT simulators* $\mathsf{Sim}_{\mathbf{C}}$ *and* $\mathsf{Sim}_{\mathbf{S}}$ *such that for all inputs* X, Y*, the following distributions are computationally indistinguishable:*

$$\left(\mathsf{View}^{\Pi}_{\mathbf{C}}(X, Y), \mathsf{Out}^{\Pi}_{\mathbf{S}}(X, Y)\right) \overset{c}{\approx} \left(\mathsf{Sim}_{\mathbf{C}}(1^\lambda, X, O_{\mathbf{C}}), O_{\mathbf{S}}\right),$$
$$\left(\mathsf{View}^{\Pi}_{\mathbf{S}}(X, Y), \mathsf{Out}^{\Pi}_{\mathbf{C}}(X, Y)\right) \overset{c}{\approx} \left(\mathsf{Sim}_{\mathbf{S}}(1^\lambda, Y, O_{\mathbf{S}}), O_{\mathbf{C}}\right).$$

3 FUZZY PROBABILISTIC DATA STRUCTURES

In this section, we introduce *Fuzzy PDS*: probabilistic data structures for approximate-membership queries. Given a query point q, the data structure returns whether q is close to some stored element under a defined distance function. We define fuzzy PDS (Defs. 8 and 9) based on the syntax of PDS given in Def. 4, and prove correctness bounds for a black-box construction given a PDS and a family of locality-sensitive hashing (LSH) functions (Def. 9).

3.1 Formalizing Fuzzy PDS

Definition 8. *Let Σ^* be an **insertion-only PRF-wrapped fuzzy PDS**, let sp^* be its setup parameters, and let F_{kPRF} be a keyed PRF. The public parameters are implicit inputs to all algorithms. Σ^* comprises six algorithms with the following syntax:*

- $(\mathsf{pp}, D) \leftarrow \Sigma^*.\mathsf{Setup}(\mathsf{sp}^*, F_{\mathsf{kPRF}}, m)$ *takes setup parameters sp^*, keyed PRF F_{kPRF} and input set size m; it outputs a data structure D and public parameters pp^*.*
- $\mathbf{u} \leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(y)$ *takes value y as input and outputs vector $\mathbf{u}$.*
- $(\mathbf{locs}, \mathbf{info}) \leftarrow \Sigma^*.\mathsf{GetInfo}(\{F_{\mathsf{kPRF}}(u)\}_{u\in\mathbf{u}})$ *takes as input a set of PRF values $\{F_{\mathsf{kPRF}}(u)\}_{u\in\mathbf{u}}$; it outputs a vector of location indices $\mathbf{locs}$ and a vector of information $\mathbf{info}$.*
- $D' \leftarrow \Sigma^*.\mathsf{Insert}(D, \mathbf{locs}, \mathbf{info})$ *takes the data structure D, vector of location indices $\mathbf{locs}$, and information $\mathbf{info}$; it returns the updated data structure D'.*
- $\mathbf{resps} \leftarrow \Sigma^*.\mathsf{Query}(D, \mathbf{locs})$ *takes the data structure D and location indices $\mathbf{locs}$; it returns a response vector $\mathbf{resps}$.*
- $b \leftarrow \Sigma^*.\mathsf{CheckMemb}(\mathbf{resps}, \mathbf{info})$ *takes values $\mathbf{resps}$ and information $\mathbf{info}$; it returns a bit $b \in \{0,1\}$.*

Correctness of an insertion-only fuzzy PDS scheme Σ^* is defined in Def. 9. We note that our definition of fuzzy PDS can be extended to support edits and deletions. But since the error probabilities of updatable PDS are not well analyzed [19], we leave this extension of our definition for future work.

Definition 9. *Let Σ^* be an insertion-only PRF-wrapped fuzzy PDS (Def. 8), and $\lambda \in \mathbb{N}$ be the security parameter. Σ^* **is** $(\rho_1, \rho_2, \delta_i, \delta_o)$-**correct** if the following holds. For every $Y \subseteq \mathcal{U}$ with $|Y| = m$, sample a PRF key $\mathsf{kPRF} \leftarrow\!\!\$\ \{0,1\}^\lambda$, run $(D, \mathsf{pp}^*) \leftarrow \Sigma^*.\mathsf{Setup}(\mathsf{sp}^*, F_{\mathsf{kPRF}}, m)$, and insert every element of Y as follows.*

For each $y \in Y$, insert the value into the PDS

$$\begin{aligned}\mathbf{u}_y &\leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(y),\\ (\mathbf{locs}_y, \mathbf{info}_y) &\leftarrow \Sigma^*.\mathsf{GetInfo}(\{F_{\mathsf{kPRF}}(u)\}_{u\in\mathbf{u}_y}),\\ D &\leftarrow \Sigma^*.\mathsf{Insert}(D, \mathbf{locs}_y, \mathbf{info}_y).\end{aligned}$$

We condition on the event that setup and all insertions succeed. Then, for every query $q \in \mathcal{U}$, let

$$\begin{aligned}\mathbf{u}_q &\leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(q),\\ (\mathbf{locs}_q, \mathbf{info}_q) &\leftarrow \Sigma^*.\mathsf{GetInfo}(\{F_{\mathsf{kPRF}}(u)\}_{u\in\mathbf{u}_q}),\\ \mathbf{resps} &\leftarrow \Sigma^*.\mathsf{Query}(D, \mathbf{locs}_q),\\ b &\leftarrow \Sigma^*.\mathsf{CheckMemb}(\mathbf{resps}, \mathbf{info}_q).\end{aligned}$$

The following two conditions must hold:

- ***Near queries.** If q is close to some stored element, i.e., if there exists $y \in Y$ such that $\mathsf{dist}(q, y) \le \delta_i$, then $\Pr[b = 1] \ge \rho_1$.*
- ***Far queries.** If q is far from every inserted element, i.e., if for all $y \in Y$ we have $\mathsf{dist}(q, y) \ge \delta_o$, then $\Pr[b = 1] \le \rho_2$.*

The probabilities are taken over the randomness of Σ^*, including the generation of D, the choice of the PRF key kPRF, and any internal randomness of GetInfo, Query, *and* CheckMemb, *conditioned on all insertions of elements in Y succeeding.*

3.2 A Generic Fuzzy PDS Construction

In this section, we give a generic construction for a fuzzy probabilistic data structure, see Fig. 3, that makes black-box use of an insertion-only PDS Σ and a family of LSH functions LSH.

Overview. The Σ^*.Setup algorithm initializes the fuzzy PDS D from its setup parameters sp^*, a keyed-PRF F_{kPRF} and m, the number of elements to be inserted. The setup parameters consist of $\mathsf{sp}_{\mathsf{PDS}}$, the parameters for the used PDS scheme, and $(\mathcal{H}, L, K)$, the parameters for the used LSH. For the LSH, $K \cdot L$ hash functions are sampled independently from the LSH family $\mathcal{H}$ where K and L describe the AND and OR constructions, respectively. The non-fuzzy PDS is initialized with Σ.Setup given $\mathsf{sp}_{\mathsf{PDS}}$, F_{kPRF}, and the number of fuzzy values to be inserted $m \cdot L$.

The Σ^*.GetFuzzyVals algorithm takes input x and computes L fuzzy values by applying the LSH AND-OR construction §2.1.

The Σ^*.GetInfo algorithm takes a set of PRF-evaluated fuzzy values $\{F_{\mathsf{kPRF}}(u)\}_{u\in\mathbf{u}}$ as input and translates them into these values' locations $\mathbf{locs}$ and auxiliary information $\mathbf{info}$ in the underlying PDS.

The Σ^*.Insert algorithm inserts the fuzzy values into D based on the output of GetInfo. To insert these values, black-box calls are made to the underlying PDS for each of the L LSH instances.

The Σ^*.Query algorithm retrieves values from the data structure at the given L locations $\mathbf{locs}$. These locations are queried in the underlying PDS and the collected responses are the output. This collection provides the information needed to verify membership.

The Σ^*.CheckMemb algorithm determines if a query matches the inserted data by aggregating membership checks across all L instances. For each instance ℓ, it invokes the underlying PDS's CheckMemb algorithm on the retrieved response $\mathbf{resps}$ and auxiliary information $\mathbf{info}$. A query is considered a match if *any* of the L instances reports membership, corresponding to the OR construction of LSH. This final aggregation balances the false negative and false positive rates determined by the LSH and PDS parameters.

Generic Correctness Bound. We now state a generic bound on the correctness of any fuzzy PDS construction (depicted in Fig. 3) and provide the proof in Appendix §B.1.

Theorem 3.1. *Let Σ^* be the insertion-only PRF-wrapped fuzzy PDS defined in Fig. 3, H be a collision-resistant hash function, and $Y \in \mathcal{U}^m$ be a set of inputs. Suppose Σ^* is instantiated with (i) a (p_1, p_2, r_i, r_o)-sensitive LSH family* LSH, *and (ii) a PRF-wrapped PDS Σ with FPP at most p_{fpp} and FNP 0. Assume that the hash functions $\{H_{\ell,k}\}_{\ell\in[L],k\in[K]}$ are sampled independently, and that each $y \in Y$ is successfully inserted into Σ^*. Then Σ^* is $(r_i, r_o, \rho_1, \rho_2)$-correct, where*

$$\rho_1 \ge 1 - (1 - p_1^K)^L \text{ and } \rho_2 \le L\big(m \cdot p_2^K + p_{\mathsf{fpp}}\big) + \mathsf{negl}(\lambda).$$

Here, we give intuition about the proof; the detailed proof is given in Appendix §B.1.

Proof sketch. In the near case, two close elements map to the same value under all K LSH functions in a partition with probability p_1^K.

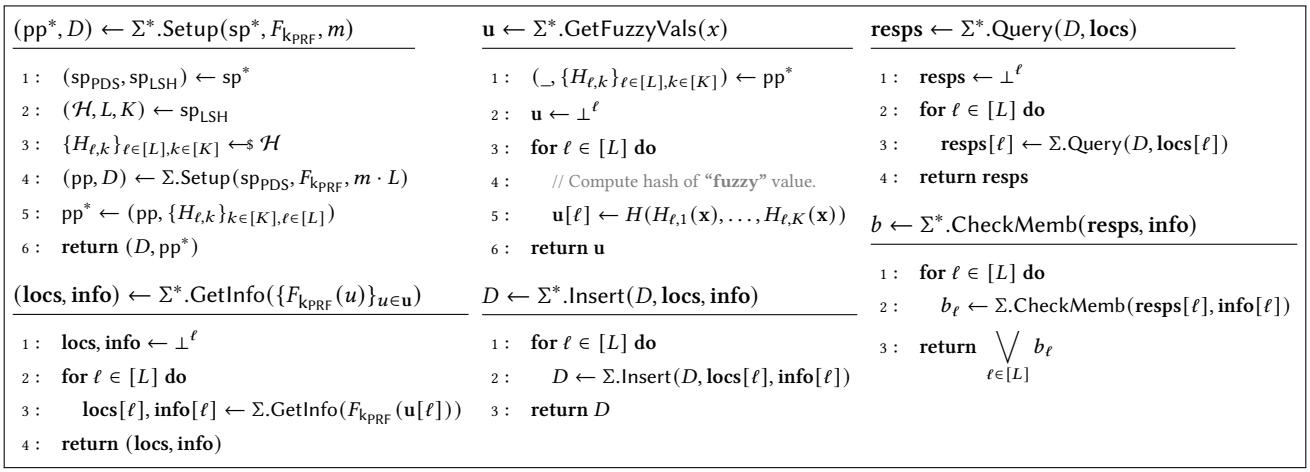

Figure 3: Pseudocode for a Fuzzy PDS Σ^* (Def. 8) with black-box use of a non-fuzzy PDS Σ (Def. 4) and LSH family $\mathcal{H}$ with parameters K, L for an AND- and OR-construction, respectively.

Thus, across L independent partitions, the probability of at least one such collision is $1 - (1 - p_1^K)^L$. In the far case, the probability that two far-away elements map to the same value for all K hash functions is at most $p_2^K + \mathsf{negl}(\lambda)$; taking a union bound over all m elements in Y and accounting for the false positive probability yields a probability of $mp_2^K + \mathsf{negl}(\lambda) + p_{\mathsf{fpp}}$. Finally, union bounding over L gives us the final inequality.

Density assumption. The bound on ρ_2 depends linearly on the database size m due to the union bound over the server elements. Ignoring the negligible term, the bound is nontrivial only when

$$L(mp_2^K + p_{\mathsf{fpp}}) < 1 \iff m < \frac{1 - Lp_{\mathsf{fpp}}}{Lp_2^K}.$$

For sufficiently dense datasets, the above bound may become vacuous even though p_2 may be smaller. This highlights a trade-off between dataset density and the false-positive guarantees provided by the fuzzy PDS.

4 FUZZY UNBALANCED PSI

In this section, we present a scalable fuzzy PSI protocol in the unbalanced, one-sided-output setting. Our protocol is obtained by combining our fuzzy PDS data structure, in a black-box manner, with techniques from unbalanced PSI [29, 42].

At a high level, LSH collisions are approximate matches, a PRF masks the outputs, and the fuzzy PDS compactly stores the server's masked LSH outputs. Using an OPRF, the client obliviously obtains their masked LSH outputs. They locally compute the locations and information within the server's fuzzy PDS. The client then obliviously retrieves the entries at these positions using PIR. Based on the retrieved values and the locally derived information, the client checks membership of their inputs. Only retrieved values matching the same (O)PRF-wrapped LSH encoding are accepted. Thus, the client learns the fuzzy intersection with the server's set, while both parties remain oblivious to each other's inputs.

4.1 Protocol Overview

Trust Assumptions and Scope. Our fuzzy unbalanced PSI protocol assumes a secure OPRF protocol Π_{OPRF} and a private single-server PIR scheme Π_{PIR} in a semi-honest setting, i.e., all parties try to learn information from an honest protocol execution. Our definition further assumes a correct insertion-only PDS Σ. We leave the extension of our scheme to malicious security, as well as the generalization to other data structures, to future work. Furthermore, our protocol makes black-box use of Π_{PIR}, hence, it can be instantiated with any secure and private PIR scheme. For simplicity, our definitions capture only single-server PIR, but we note that the definitions can be trivially extended to accommodate multi-server PIR.

Offline phase. This setup can be performed once and reused across queries. In the offline phase, the server runs the setup for the fuzzy PDS. It first samples a PRF key $\mathsf{k}_{\mathsf{PRF}}$ and uses the setup parameters sp^* and its set Y of size m to run $\Sigma^*.\mathsf{Setup}(\mathsf{sp}^*, F_{\mathsf{k}_{\mathsf{PRF}}}, m)$. This returns the empty fuzzy PDS D and public parameters pp^*. Here, pp^* contains the public LSH descriptions and any parameters of the underlying PDS. Next, the server inserts its points into the fuzzy PDS. For every point $y \in Y$, it computes the fuzzy values as

$$\mathbf{u} \leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(y).$$

This is done by applying the LSH functions to y. The server then evaluates the PRF on each value in $\mathbf{u}$ and derives the corresponding PDS locations and auxiliary information by computing

$$(\mathbf{locs}, \mathbf{info}) \leftarrow \Sigma^*.\mathsf{GetInfo}(\{F_{\mathsf{k}_{\mathsf{PRF}}}(u)\}_{u\in\mathbf{u}}).$$

Finally, it inserts this information into the data structure:

$$D \leftarrow \Sigma^*.\mathsf{Insert}(D, \mathbf{locs}, \mathbf{info}).$$

Intuitively, the PRF hides the raw LSH outputs, and $\Sigma^*.\mathsf{GetInfo}$ outputs the locations and auxiliary information for the fuzzy PDS. The offline phase includes the PIR offline setup if needed.

Online phase. The client initializes the intersection $I = \emptyset$ and processes each point $x \in X$. It computes the fuzzy values $\mathbf{v}$ associated with x by running $\Sigma^*.\mathsf{GetFuzzyVals}(x)$, the same algorithm used by

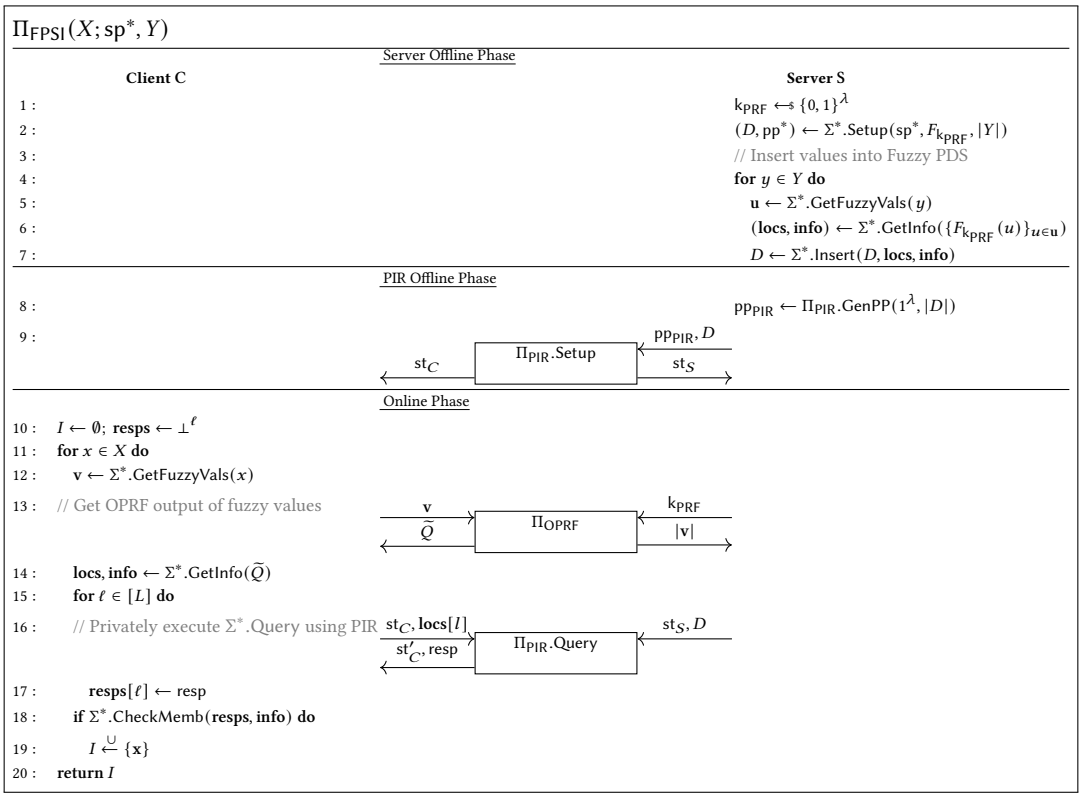

Figure 4: Our fuzzy PSI protocol Π_{FPSI} built using a Fuzzy PDS Σ^*, OPRF Π_{OPRF} and PIR scheme Π_{PIR}. Public parameters are implicit inputs to the corresponding protocols.

the server. To hide these LSH values from the server while keeping the server's PRF key private, the client and server obliviously execute the batched PRF on the elements of $\mathbf{v}$. At the end of the OPRF execution, the client learns $\widetilde{Q} = \{F_{\mathsf{k}_{\mathsf{PRF}}}(v)\}_{v \in \mathbf{v}}$ without learning $\mathsf{k}_{\mathsf{PRF}}$. The client then locally computes the fuzzy PDS locations **locs** and the auxiliary information **info** needed to later check whether the retrieved responses constitute a valid fuzzy match by:

$$(\mathbf{locs}, \mathbf{info}) \leftarrow \Sigma^*.\mathsf{GetInfo}(\widetilde{Q}).$$

While the client could run the fuzzy PDS query algorithm Query locally, this would require downloading the entire fuzzy PDS. To avoid this significant bandwidth overhead, the client uses PIR to privately retrieve the entries at the locations **locs**. In particular, for each loc $\in$ **locs**, the client and server jointly execute $\Pi_{\mathsf{PIR}}.\mathsf{Query}$ on index loc, the client obtains resp and appends it to the response vector **resps**. The server learns nothing about the queried locations.

Finally, the client locally checks whether the retrieved responses constitute membership in the fuzzy PDS. For each point $x \in X$, after obtaining the PIR output **resps**, the client invokes

$$b \leftarrow \Sigma^*.\mathsf{CheckMemb}(\mathbf{resps}, \mathbf{info}).$$

If this check accepts (i.e., $b = 1$), the client adds x to the output set I. Thus, x is output whenever its fuzzy values are consistent with entries inserted by the server.

A note on spatial hashing. One can replace our use of LSH in the fuzzy PDS with spatial hashing to obtain a protocol for, e.g., the L_∞ norm. See Appendix C.1 for a more detailed discussion. We leave investigating the concrete performance of such an approach for future work.

4.2 Complexity

We now state the complexities of our generic fuzzy PSI protocol in Fig. 4. The line numbers in this section refer to Fig. 4. We consider the offline phase (i.e., preprocessing and setup cost) separately from the online query cost, since the offline phase is a one-time cost that can be amortized over all online phases.

The functions $\mathtt{BW}$ and $\mathtt{Comp}$ report an algorithm's communication and computation cost, respectively, while a subscript on these functions specifies the party, **S** or **C**, in an interactive algorithm.

Communication. We assume public knowledge of the public parameters. Depending on the chosen PIR scheme Π_{PIR}, the offline phase (line 9) might require interaction, and hence communication, between client and server. Thus, $\mathtt{BW}(\Pi_{\mathsf{PIR}}.\mathsf{Setup})$ is different per PIR scheme and depends on security parameter λ and the size $|D|$. In the online phase, the client and server interactively run nL OPRF calls and $n|\mathbf{v}|$ PIR queries, where $|\mathbf{v}| = L$.

The total communication cost for our generic construction is $\mathtt{BW}(\Pi_{\mathsf{PIR}}.\mathsf{Setup}) + nL\,(\mathtt{BW}(\Pi_{\mathsf{OPRF}}) + \mathtt{BW}(\Pi_{\mathsf{PIR}}.\mathsf{Query}))$. Additionally,

some OPRF schemes allow preprocessing in the offline phase to improve the online evaluation cost [4].

Computation. In the offline phase, the server sets up the fuzzy PDS Σ^*.Setup, inserts all m elements using $K \cdot L$ LSH functions, and executes Π_{PIR}.Setup with the client.

In the online phase, the server interactively runs the OPRF evaluation and Π_{PIR}.Query once for the $L \cdot n$ client queries. If an OPRF with preprocessing is used, this computational cost is moved to the offline phase for both parties.

This results in a total server computation cost of $\mathsf{Comp}(\Sigma^*.\mathsf{Setup})+\mathsf{Comp}_{\mathsf{S}}(\Pi_{\mathsf{PIR}}.\mathsf{Setup})+m(\mathsf{Comp}(\Sigma^*.\mathsf{GetFuzzyVals})+\mathsf{Comp}(\Sigma^*.\mathsf{GetInfo})+\mathsf{Comp}(\Sigma^*.\mathsf{Insert}))+nL(\mathsf{Comp}_{\mathsf{S}}(\Pi_{\mathsf{OPRF}})+\mathsf{Comp}_{\mathsf{S}}(\Pi_{\mathsf{PIR}}.\mathsf{Query}))$.

The client computation cost in the offline phase only includes the PIR setup. In the online phase, the client runs the interactive OPRF and PIR query algorithms, and locally executes Σ^*.GetFuzzyVals for all n values in X, as well as Σ^*.GetInfo and Σ^*.Query for all nL fuzzy values in X.

This results in a total client computation cost of $\mathsf{Comp}_{\mathsf{C}}(\Pi_{\mathsf{PIR}}.\mathsf{Setup})+n\cdot(\mathsf{Comp}(\Sigma^*.\mathsf{GetFuzzyVals})+\mathsf{Comp}(\Sigma^*.\mathsf{GetInfo}))+nL\cdot(\mathsf{Comp}_{\mathsf{C}}(\Pi_{\mathsf{OPRF}})+\mathsf{Comp}(\Sigma^*.\mathsf{Query})+\mathsf{Comp}_{\mathsf{C}}(\Pi_{\mathsf{PIR}}.\mathsf{Query}))$.

4.3 Correctness and Security

Π_{FPSI} is correct if Σ^*, the OPRF and PIR schemes are correct. We formally state and prove this in Appendix C.

Theorem 4.1 (Semi-honest Server). *Let Π_{PIR} be a PIR scheme (Def. 5), Π_{OPRF} be a batch OPRF scheme (Def. 3), and Σ^* be an insertion-only PRF-wrapped fuzzy PDS (Def. 8). Then protocol Π_{FPSI} in Fig. 4 securely realizes $\mathcal{F}_{\mathsf{FPSI}}$ (Fig. 2) against a semi-honest server.*

Proof sketch. By the security of Π_{OPRF}, the server learns only an upper bound on the input set size and nothing about the client's input values. Similarly, by the privacy of Π_{PIR}, the server does not learn which buckets the client retrieves. Thus, the server learns an upper bound on the client's input size.

Theorem 4.2 (Semi-honest Client). *Let F be a secure PRF and let Π_{OPRF} be a secure OPRF scheme (Def. 3) satisfying the pseudorandomness property. Let Σ^* be an insertion-only PRF-wrapped fuzzy PDS (Def. 8). Then the protocol Π_{FPSI} in Fig. 4 securely realizes $\mathcal{F}_{\mathsf{FPSI}}$ (Fig. 2) against a semi-honest client.*

Proof sketch. Similarly, by the security of Π_{OPRF}, the batch OPRF protocol does not reveal any information about the server's PRF key to the client. Moreover, since the server's elements are inserted into the PRF-wrapped fuzzy PDS, the elements in the database are indistinguishable from random, and thus the client learns nothing about the plaintext values in the database beyond those that their own values match.

The proofs of the above security theorems can be found in Appendices C.3 and C.4, respectively.

5 EVALUATION

This section describes the implementation of our fuzzy PSI protocol and evaluates its performance compared to related work [14, 49, 58]. We generally refer to the two parties as the client **C** and the server **S**, holding the sets X and Y, respectively, where $|X| \ll |Y|$. We examine two real-world use cases: private matching of *location data* (§5.2) and *machine learning (ML) labels* (§5.3). We report the FPP as $FP/(FP+TN)$ and the FNP as $FN/(TP+FN)$, expressed as percentages. All experiments were conducted with an AMD Ryzen 7970X processor (32 cores/64 threads) and 256 GB of RAM.

5.1 Implementation

Our fuzzy PSI protocol Π_{FPSI} makes black-box use of its building blocks and can hence accommodate different design choices for different use cases. We instantiate and optimize our fuzzy PSI Π_{FPSI}, with a cuckoo filter (CF) [18] for Σ, BatchPIR [47] for Π_{PIR} and GC-AES, an OPRF based on garbled circuits (GC) for oblivious AES evaluations [4], for Π_{OPRF}.

For the LSH, we use E2LSH [15] for the L_2 distance. E2LSH defines a family of $K \cdot L$ hash functions, each parameterized by a random vector $\mathbf{a} \in \mathbb{R}^d$, where d is the dimension of $\mathbf{x}$, and a random offset $b \in [0, w]$, where $w > 0$ is a fixed width parameter: $h_{\mathbf{a},b}(\mathbf{x}) = \left\lfloor \frac{\langle \mathbf{a},\mathbf{x}\rangle + b}{w} \right\rfloor$. For each hash function, $\mathbf{a}$ and b are sampled independently. We optimize the parameters of E2LSH on a hold-out set of the used datasets by performing a grid search over w, K, and L, in order to achieve the target FPP and FNP. Further evaluation of our protocol with another LSH function and PIR scheme can be found in §5.4.

5.2 Application: Location Data

This section evaluates the performance of our protocol and related work for the use case of private matching of two-dimensional GPS location data. As a motivating application, consider regional outbreak monitoring (e.g., mosquito-borne diseases), where a server **S** holds locations of infected individuals and a client **C** wishes to learn whether they have been close to someone with this disease, indicating they might have been bitten by the same mosquito. To assess risk, **C** should learn which of its locations matched, without learning information about non-matching locations in Y, while **S** should learn nothing about **C**'s location history. In our experiments, two locations are considered "close" if their Euclidean distance[1] is at most 0.09 (≈ 9 km) and "far" if it is more than 0.1 (≈ 10 km). In other words, following Def. 2, we set $r_i = 0.09$ and $r_o = 0.1$.

Experiment Design. We use the Gowalla dataset [50] containing approximately 6.5 million locations from check-ins of a social network website. For experiments requiring larger datasets (e.g., Tab. 1 which scales up to 1B data points), we synthetically generate additional locations by randomly jittering existing check-in coordinates within a radius of approximately 250 meters. We first hold out 20% of the dataset for parameter optimization. The remaining points are split into disjoint server and client pools, containing 70% and 30% of the points, respectively, from which elements of **S** and **C** are randomly sampled. Half of **C**'s points are close to points in **S**, i.e., there exists $x \in X, y \in Y$ with $\|x - y\|_2 \leq 0.09$. The other half are far from all of **S**'s points.

Performance of Π_{FPSI} with error. Fig. 5 displays the total runtime and communication cost of our protocol, when configured to achieve the minimum possible FPP (shown in red text), while maintaining FNP of < 1%.

[1] For the small geographic thresholds that we use, the Earth curvature is negligible and coordinates are treated as planar.

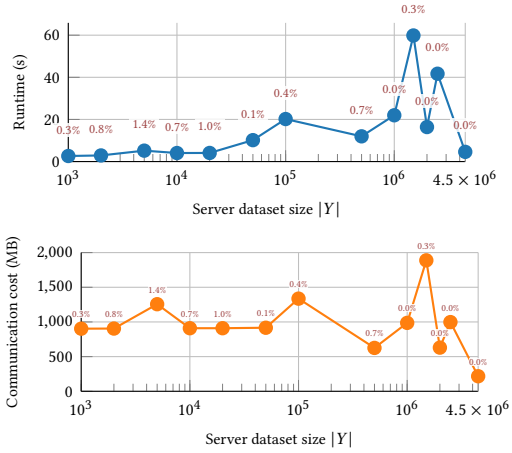

Figure 5: Performance at minimum FPP (in red) at FNP < 1%.

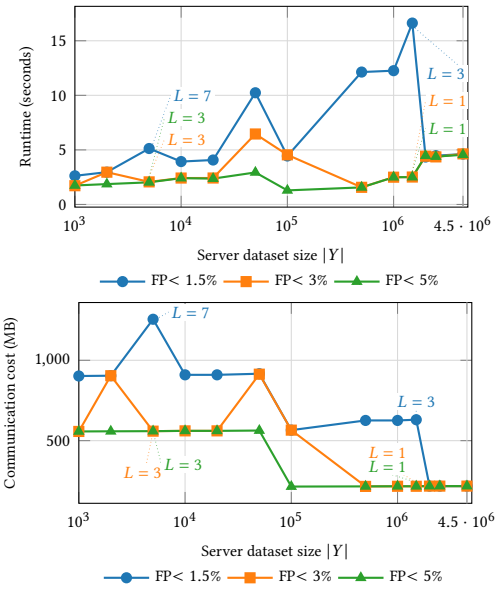

Figure 6: Performance of Π_{FPSI} for different target FPP (FNP < 1%). L values highlight their impact on performance.

Fig. 5 shows that both runtime and communication overhead remain reasonable while achieving low FPP, showing that our approach provides good performance with a reasonable accuracy loss as trade-off. While the cost of our protocol is dominated by $|Y|$, the size of the server set, the runtime and communication cost fluctuate. This is due to the parameter optimization resulting in different values for L and the size of the CF.

The performance's dependence on L is better revealed in Fig. 6, which shows the cost under different FPP targets (while FNP < 1%). As FPP becomes less strict, the performance cost is mostly decreased. The L values explain the performance differences. Stricter FPP rates require larger K and smaller w, reducing collision probabilities but increasing FNP. To keep FNP below 1%, L has to be increased, resulting in higher runtime and communication costs. In other words, allowing larger FPP rates can reduce the overall cost by enabling smaller values of L. It is worth noting, however, that the achievable error rates depend on the dataset. In some cases, $L = 1$ (i.e., the minimum possible value) is already sufficient to achieve an FPP rate below 1%. In such scenarios, relaxing the FPP target does not provide additional performance improvements.

We further see that larger datasets generally require a smaller L. As Y contains more points, the probability that multiple points already collide with those in X statistically increases, reducing the need for a larger L. This, in turn, improves performance. For instance, Fig. 6 shows that $L = 1$ for all $|Y| > 10^6$ for FPP<3%.

For $|X| = 2^{10}$ and $|Y| = 1$ billion, Tab. 1 shows that Π_{FPSI} with BatchPIR takes ≈ 12.4 minutes with < 3% FPP and < 1% FNP rates.

Cost of C and S. Fig. 8 presents the runtime breakdown between the client (C) and the server (S) for a target FPP below 3% and FNP below 1%. A significant fraction of the total runtime is spent in the offline phase (e.g., nearly half of the total runtime when $|Y| = 4.5$ million). However, we note that this cost can be amortized across multiple queries or clients. Furthermore, the client-side runtime remains low for all $|Y|$, demonstrating the applicability of our protocol to settings where the client has limited computational power (e.g., a smartphone) and the server can handle higher computational costs (e.g., a cluster). Note that the runtime depends on the chosen parameters for Π_{FPSI} and the given input sets, as previously discussed. Hence, fluctuations are visible with different set sizes.

Comparison to Related Work. We compare our protocol against three state-of-the-art fuzzy PSI schemes for L_2 distance [14, 49, 58]. To the best of our knowledge, these are the only existing fuzzy PSI protocols supporting the L_2 distance. These protocols have a negligible error probability (2^{-40}), but do not consider the unbalanced setting, highlighting the gap that our work addresses. We evaluate these protocols using their public prototypes, but since these implementations are not designed for very large datasets, some experiments either crashed or failed to complete within eight hours. We mark failed experiments with $*$ for $|Y| \leq 2^{22}$ and omit the corresponding lines due to space constraints for $|Y| \geq 2^{24}$.

Another related unbalanced fuzzy PSI protocol is that of Gao et al. [21], based on fuzzy mapping: the inputs are first mapped to identifiers to identify candidate pairs, which are then filtered using a cryptographic fuzzy-matching protocol. Although [21] reports experimental results for L_2, its available implementation does not provide code for the L_2 instantiation; we therefore exclude it from Fig. 7. Nevertheless, [21] report a runtime of 855 s and 1.6 GB of communication for $|X| = 2^8$ and $|Y| = 2^{16}$. In comparison, Π_{FPSI} requires only 3 s and 216 MB of communication even for the substantially larger setting of $|X| = 2^{10}$ and $|Y| = 2^{20}$. This highlights the trade-off between lower computational cost and small error that our construction provides over fuzzy-matching.

Fig. 7 shows that while our protocol incurs higher runtime and communication costs on smaller sets, it outperforms the other protocols as the server set size grows. For example, with $|Y| = 4.5 \cdot 10^6$, our protocol is nearly two orders of magnitude faster and an order of magnitude cheaper in communication.

Tab. 1 further shows the runtime under different network throughputs for both $|X| = 2^{10}$ and $|X| = 2^{13}$, with the best results marked

Table 1: Performance comparison of our protocol (Π_{FPSI}, < 3% FPP, <1% FNP) with prior work. Runtime is given in seconds and communication cost in MB. Best results are highlighted in green; omitted related work lines indicate failed execution.

$\lvert X\rvert$	$\lvert Y\rvert$	Scheme	Comm. (MB)	Runtime (s) LAN	200 Mbps	50 Mbps	5 Mbps
2^{10}	2^{20}	**Ours**	216	3	11	37	349
		[49]	893	400	436	543	1 829
		[58]	468	82	100	157	830
		[14]	18 250	17 055	17 785	19 975	46 255
	2^{22}	**Ours**	217	5	13	39	352
		[49]	4 019	1 830	1 990	2 472	8 260
		[58]	2 106	346	431	683	3 716
		[14]	82 100	76 750	80 034	89 886	208 110
	2^{24}	**Ours**	659	57	84	163	1 112
	2^{26}	**Ours**	2 985	944	1 063	1 422	5 720
	2^{30}	**Ours**	720	744	773	860	1 901
2^{13}	2^{20}	**Ours**	1 472	11	70	246	2 365
		[49]	895	403	439	546	1 835
		[58]	480	136	155	213	904
	2^{22}	**Ours**	1 475	12	71	248	2 371
		[49]	4 029	1 819	1 980	2 463	8 264
		[58]	2 166	408	494	754	3 873
	2^{24}	**Ours**	4 512	180	360	902	7 398
	2^{26}	**Ours**	4 550	321	503	1 049	7 601
	2^{30}	**Ours**	5 010	1 137	1 338	1 939	9 151

in green. This scalability comes at the cost of a small error (in this evaluation: FPP < 3% and FNP < 1%), whereas the competing protocols have negligible errors. Nevertheless, accepting this small error makes fuzzy PSI practical at scales that were previously infeasible. In contrast, for smaller datasets (e.g., $|Y| < 10^5$), [49] and [58], which focus on small-scale settings, can achieve lower runtimes.

Comparison with exact PSI. We note that PSI with exact matching is not applicable in this use case, as X and Y do not hold identical records. For example, in our 1-billion experiment, where we augment the Gowalla dataset with additional jittered values, the two sets have 0 exactly matching records. Thus, exact PSI would return an empty intersection. Moreover, using exact PSI for fuzzy matching over a neighborhood of radius δ would require enumerating the neighborhood around each client record. This would result in a computational blow-up on the order of δ^d, for d dimensions.

5.3 Application: Machine Learning Class Labels

In § 5.2, we focus on large server datasets and low-dimensional data, showing that Π_{FPSI} maintains reasonable performance even at scales that are beyond existing fuzzy PSI protocols. In this section we show that Π_{FPSI} can address another important gap: efficient fuzzy PSI for high-dimensional data. Among the protocols compared in §5.2, [49] was designed specifically for two-dimensional L_2 data, while [58] discusses a high-dimensional variant but without providing an implementation. Prefix Representation [14] supports high-dimensional data but it remains impractical for higher dimension, as shown in the following.

Motivation. We are motivated by the use case of Federated Learning [45, 54] where two parties, **C** and **S**, want to jointly train an ML model. To do so, each party first trains a model locally on its own data, and then shares the trained model with a central entity. This entity will then combine them. **C** wishes to know how many of its elements are similar to those of **S**. The motivation behind this is two-fold. First, **C** wants to avoid using records that are already represented in **S**'s dataset, in order to reduce redundancy and unnecessary training cost. Second, **C** would like to assess whether its dataset aligns with **S**'s, in order to understand if the collaboration is beneficial. If **C** holds a significantly different dataset, then collaborating with **S** might be less effective, or additional ML techniques may be required when combining the models to ensure that **C** is well represented in the final model and any distributional biases are mitigated. Note that exact PSI is insufficient in this use case, as it is unlikely that **S** and **C** have exactly the same training points. Instead, it is more reasonable to assume that even when training examples are related to the same individual (e.g., a patient), they may slightly differ (e.g., due to noise or small changes in the patient's results).

Experiment Design. This experiment fits the use case of comparing medical images of two institutions. Due to ethical concerns with medical datasets, we instead use image datasets commonly used in the field of ML. Concretely, we use the MNIST [16] and Fashion-MNIST [63] datasets, which both contain 28×28 grayscale images: MNIST contains handwritten digits (0-9), while Fashion-MNIST contains clothing items (10 classes).

To evaluate our approach, we first need to define what it means for a training point to be "close" or "far" to another point. Aumüller et al. [2] have precomputed the Euclidean distances for the 100 nearest neighbors of each data point in the test set of MNIST and Fashion-MNIST. We focus on estimating how many elements from the same class X and Y share. Since it is possible that two elements belong to the same class but have a large L_2 distance (i.e., are outliers), we additionally require elements to be sufficiently close in terms of L_2 distance. Using the intra-class and inter-class statistics of the datasets, we assume that two same-class elements are close if their L_2 distance is at most 1000, while two different-class elements are far if their Euclidean distance is at least 2000. In other words, we define two elements x, y as *close* if $x_{\text{class}} = y_{\text{class}}$ and $||x - y||_2 \leq 1000$, and *far* if $x_{\text{class}} \neq y_{\text{class}}$ and $||x - y||_2 \geq 2000$.

S randomly samples elements from the test set which belong in the first five classes which means that it has ≈ 5000 elements[2]. Similarly to § 5.2, for **C** we randomly sample elements such that half of its dataset are "close" elements and half are "far".

Accuracy. Fig. 9 shows the minimum FPP/FNP percentages as the size of the client's dataset $|X|$ increases (in contrast to § 5.2, where we scale $|Y|$). In all cases, the FP/FN rates remain relatively stable and low, suggesting that the accuracy of Π_{FPSI} is largely unaffected by $|X|$. Thus, the experiment shows that Π_{FPSI} can maintain acceptable error rates even for high-dimensional image data.

This setting is more challenging than that of location data for two reasons. First, the high dimensionality of the data ($d = 784$ for both datasets) may make collisions less likely, as points become more dispersed in the feature space and random projections are less

[2]To the best of our knowledge, no larger ML dataset with precomputed Euclidean distances is available beyond MNIST/Fashion-MNIST [2].

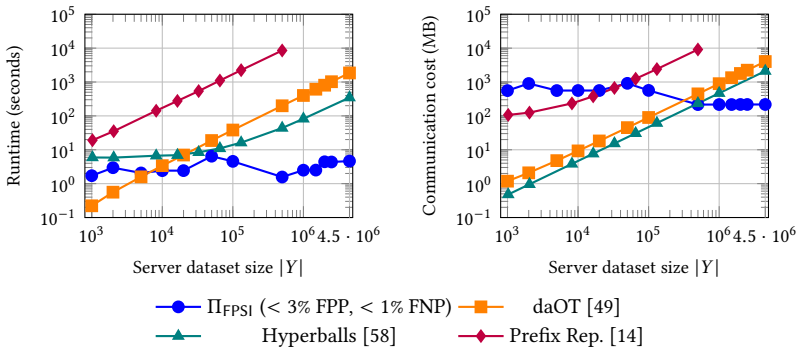

Figure 7: Performance comparison with client's $|X| = 1024$ (log scale).

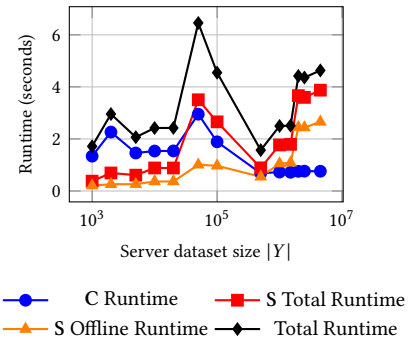

Figure 8: Runtime of server S and client C with FPP< 3%, FNP< 1% (log scale).

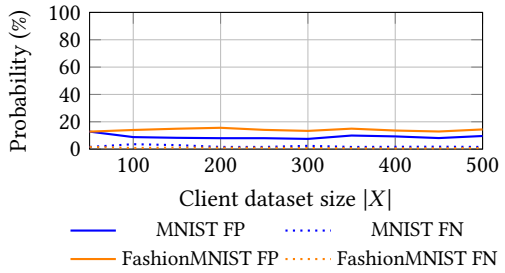

Figure 9: Error of Π_{FPSI} on high-dimensional ML datasets.

likely to map similar points to the same bucket. Second, parameter calibration on the hold-out set is more difficult because the dataset is relatively small, limiting the number of points available for tuning.

Performance. Π_{FPSI} also achieves reasonable performance for high-dimensional data. For $|X| = 500$ and $|Y| = 5000$, Π_{FPSI} takes 31 s on MNIST with 2259 MB of communication. For the same dataset sizes on Fashion-MNIST, Π_{FPSI} takes 17 s with 1748 MB of communication. Prefix Rep. [14] could theoretically be applied in this setting as well. However, it remains largely impractical: for $|X| = |Y| = 512$, [14] already takes 10 minutes to run and requires 27.3 GB of communication[3]. Recall that even for low-dimensional data, Fig. 7 shows that [14] is slower than Π_{FPSI}. The other protocols ([49, 58]) are inapplicable for high dimensions.

5.4 Implementation Extensions & Additional Use Cases

Large and High-Dimensional Data. While §5.2 focused on a large, low-dimensional dataset and §5.3 on small, high-dimensional datasets, we now evaluate the setting where the data is both large and high-dimensional. We use SpaceV [53], a real-world semantic vector-search benchmark with 100-dimensional vectors and defined ground-truth nearest (wrt L_2) neighbors. We set $r_i = 30$, $r_o = 80$, $|X| = 2^{10}$, and $|Y| = 2^{20}$. Π_{FPSI} achieves an FPP of 2% and an FNP of 2%, with a runtime of 49.4 minutes and 9.8 GB of communication using BatchPIR.

[3]This is the largest dataset size for which we could report results for [14], as larger sizes exhausted the available memory.

An advantage of BatchPIR is that it does not require any additional trust assumptions. Other PIR schemes achieve notably better performance by introducing the assumption of two non-colluding PIR servers. For instance, ChecklistPIR [36] reduces the runtime of Π_{FPSI} to $\approx$ 18 minutes (with the same FPP/FNP rates), requiring 6.7 GB of communication, but assumes two non-colluding PIR servers.

Feature Representation. Our approach can remain effective even for data that is not naturally represented in L_2 space. To evaluate this, we use a standard publication record linkage dataset containing bibliographic records from DBLP and ACM [37]. We consider two organizations holding publication libraries, e.g., ACM (C) and DBLP (S), that wish to find the publications they have in common without revealing their whole collections. Note that fuzzy matching is necessary to handle common inconsistencies in the publications (e.g., errors, missing fields).

Since the records are not naturally embedded in an L_2 space, we construct a fixed-dimensional feature representation for each publication by converting its bibliographic information into a 28-dimensional feature vector. The embedding is designed so that bibliographically similar records are mapped to nearby (wrt L_2) points, making the Euclidean distance a meaningful proxy for record similarity. In other words, the role of the embedding is to provide an L_2 representation suitable for *E2LSH*.

As the dataset of [37] is relatively small, we set $|Y| = 2600$ and $|X| = 128$. Π_{FPSI} achieves an FPP of 3% and an FNP of 3% (precision = 0.96, recall = 0.96) in 15.8 s while requiring 619 MB of communication. Note that the accuracy and error rates are computed directly against the ground truth provided by [37], rather than by comparing the embeddings. Among the compared protocols, as discussed, only [14] could be executed in higher dimensions. However, we were unable to evaluate [14] on the target dataset size ($|Y| = 2600$) as the implementation crashed during execution. Nevertheless, even for the much smaller setting of $|X| = |Y| = 128$, [14] already required 260 MB of communication and 6 s of runtime. Because such feature representations are often moderately or highly dimensional, the linear dependence of our instantiation of Π_{FPSI} on the dimensionality (through E2LSH) makes the protocol a particularly well suited approach. In conclusion, this experiment shows that even data that is not inherently suited to the Euclidean distance metric

can be transformed into an appropriate fixed-dimensional feature representation, enabling the use of Π_{FPSI}.

Beyond L_2. Throughout the paper, we use the L_2 distance, but our modular design supports other metrics with an appropriate LSH. To show this, we repeat the experiment using *edit distance* on the concatenated title and year of each record and OrderMinHashing [44] as the LSH function. Note that we exclude authors because their semantics are not well captured by *edit distance*; for example, reordering the same author list can substantially increase the distance even though it represents the same record. We obtain an FPP of 1% and FNP of 4% (precision = 0.98, recall = 0.95) in 7.7 s with 318 MB of communication.

6 DISCUSSION

Limitations. We trade off a small non-negligible FP and FN error in exchange for faster runtimes. Our protocol fills a gap in the design space, enabling significant scalability and efficiency gains for use cases that can tolerate a low tunable inaccuracy. However, this makes our scheme non-applicable when any error is unacceptable.

Moreover, we assume that the E2LSH parameters can be tuned with an auxiliary distribution. In our experiments, we used a hold-out part of the datasets. Tuning the parameters directly on the private data might leak certain characteristics of the distribution. For example, larger values of K and L in E2LSH may indicate that matching is harder because the dataset is sparse. In practice, public datasets (e.g., public facial images) can be used for sensitive tasks (e.g., face recognition). Furthermore, the choice of hyperparameters can affect the protocol's runtime. Since different datasets may require different choices of K, L, and w in E2LSH, the protocol's runtime ultimately depends on the underlying distribution of the dataset. For example, in § 5, we showed how increasing L directly impacts the runtime and the communication cost. In particular, for less dense datasets, we can choose larger w without compromising the error rate. Similarly, we can select smaller K and larger L when the database is sparse. Thus, the parameter choices may leak some information about the sparsity of the underlying database. Finally, the number of PIR requests a client makes to the server might reveal information about the client's set size; this can be addressed by padding the number of requests to a specified, known value.

Conclusion. In this work, we present a scalable fuzzy PSI protocol for the unbalanced setting. The core of our protocol is a generalized data structure that combines LSH with compact filters to support fuzzy membership queries. We integrate this data structure with OPRF and PIR techniques from the unbalanced PSI literature to obtain a practically efficient fuzzy PSI protocol for large server-side datasets, with client costs that can scale sublinearly in the server set size at the cost of a small, tunable error rate.

Future work. This work leaves open several interesting directions for future work, including evaluating the performance of our fuzzy PDS with other PDS instantiations, such as XOR filters and Binary Fuse Filters. Another interesting question is whether the error rate can be further reduced without compromising efficiency. Finally, investigating the aforementioned potential leakage of the selected hyperparameters of E2LSH is an interesting direction.

ACKNOWLEDGMENTS

This work was supported by the Zurich Information Security and Privacy Center (ZISC) and the European Commission under the project RECITALS (101168490). We thank Daniela Thurner for her contributions to early explorations of this topic.

REFERENCES

[1] [n.d.]. Probabilistic: Probabilistic data structures in Redis. https://redis.io/docs/latest/develop/data-types/probabilistic/. Accessed: 2026-06-13.
[2] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374. https://doi.org/10.1016/j.is.2019.02.006
[3] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2011. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *ACM CCS 2011*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM Press, 691–702. https://doi.org/10.1145/2046707.2046785
[4] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient Garbling from a Fixed-Key Blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy (SP '13)*. IEEE Computer Society, USA, 478–492. https://doi.org/10.1109/SP.2013.39
[5] Arindam Bhattacharya, Chathur Gudesa, Amitabha Bagchi, and Srikanta Bedathur. 2022. New Wine in an Old Bottle: Data-aware Hash Functions for Bloom Filters. *Proc. VLDB Endow.* 15, 9 (2022), 1924–1936.
[6] Burton H. Bloom. 1970. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM* 13, 7 (July 1970), 422–426. https://doi.org/10.1145/362686.362692
[7] Dung Bui, Gayathri Garimella, Peihan Miao, and Van Long Phuoc Pham. 2025. New Framework for Structure-Aware PSI From Distributed Function Secret Sharing. In *ASIACRYPT 2025, Part V (LNCS)*, Goichiro Hanaoka and Bo-Yin Yang (Eds.), Vol. 16249. Springer, Singapore, 294–326. https://doi.org/10.1007/978-981-95-5116-3_10
[8] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. 2022. SoK: Oblivious Pseudorandom Functions. In *2022 IEEE European Symposium on Security and Privacy*. IEEE Computer Society Press, 625–646. https://doi.org/10.1109/EuroSP53844.2022.00045
[9] Sofía Celi and Alex Davidson. 2024. Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries. In *ACM CCS 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM Press, 4107–4121. https://doi.org/10.1145/3658644.3670271
[10] Anrin Chakraborti, Giulia Fanti, and Michael K. Reiter. 2023. Distance-Aware Private Set Intersection. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 319–336. https://www.usenix.org/conference/usenixsecurity23/presentation/chakraborti-intersection
[11] Justin Chan, Landon P. Cox, Dean P. Foster, Shyam Gollakota, Eric Horvitz, Joseph Jaeger, Sham M. Kakade, Tadayoshi Kohno, John Langford, Jonathan Larson, Puneet Sharma, Sudheesh Singanamalla, Jacob E. Sunshine, and Stefano Tessaro. 2020. PACT: Privacy-Sensitive Protocols And Mechanisms for Mobile Contact Tracing. *IEEE Data Eng. Bull.* 43, 2 (2020), 15–35. http://sites.computer.org/debull/A20june/p15.pdf
[12] Hao Chen, Ilaria Chillotti, Yihe Dong, Oxana Poburinnaya, Ilya P. Razenshteyn, and M. Sadegh Riazi. 2020. SANNS: Scaling Up Secure Approximate k-Nearest Neighbors Search. In *USENIX Security 2020*, Srdjan Capkun and Franziska Roesner (Eds.). USENIX Association, 2111–2128. https://www.usenix.org/conference/usenixsecurity20/presentation/chen-hao
[13] Lukasz Chmielewski and Jaap-Henk Hoepman. 2008. Fuzzy Private Matching (Extended Abstract). In *ARES*. IEEE Computer Society, 327–334.
[14] Chengrui Dang, Xv Zhou, and Bei Liang. 2025. Efficient Fuzzy PSI Based on Prefix Representation. In *ACM CCS 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 2204–2218. https://doi.org/10.1145/3719027.3765203
[15] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, Jack Snoeyink and Jean-Daniel Boissonnat (Eds.). ACM, 253–262. https://doi.org/10.1145/997817.997857
[16] Li Deng. 2012. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Process. Mag.* 29, 6 (2012), 141–142.
[17] Thai Duong, Duong Hieu Phan, and Ni Trieu. 2020. Catalic: Delegated PSI Cardinality with Applications to Contact Tracing. In *ASIACRYPT 2020, Part III (LNCS)*, Shiho Moriai and Huaxiong Wang (Eds.), Vol. 12493. Springer, Cham, 870–899. https://doi.org/10.1007/978-3-030-64840-4_29
[18] Bin Fan, David G. Andersen, Michael Kaminsky, and Michael D. Mitzenmacher. 2014. Cuckoo Filter: Practically Better Than Bloom. In *Proceedings of the 10th ACM*

International on Conference on Emerging Networking Experiments and Technologies (CoNEXT). 75–88. https://doi.org/10.1145/2674005.2674994

[19] Mia Filic, Keran Kocher, Ella Kummer, and Anupama Unnikrishnan. 2024. Deletions and Dishonesty: Probabilistic Data Structures in Adversarial Settings. In *ASIACRYPT 2024, Part IV (LNCS)*, Kai-Min Chung and Yu Sasaki (Eds.), Vol. 15487. Springer, Singapore, 137–168. https://doi.org/10.1007/978-981-96-0894-2_5

[20] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. 2004. Efficient Private Matching and Set Intersection. In *EUROCRYPT 2004 (LNCS)*, Christian Cachin and Jan Camenisch (Eds.), Vol. 3027. Springer, Berlin, Heidelberg, 1–19. https://doi.org/10.1007/978-3-540-24676-3_1

[21] Ying Gao, Lin Qi, Xiang Liu, Yuanchao Luo, and Longxin Wang. 2024. Efficient Fuzzy Private Set Intersection from Fuzzy Mapping. In *ASIACRYPT 2024, Part VI (LNCS)*, Kai-Min Chung and Yu Sasaki (Eds.), Vol. 15489. Springer, Singapore, 36–68. https://doi.org/10.1007/978-981-96-0938-3_2

[22] Gayathri Garimella, Benjamin Goff, and Peihan Miao. 2024. Computation Efficient Structure-Aware PSI from Incremental Function Secret Sharing. In *CRYPTO 2024, Part VIII (LNCS)*, Leonid Reyzin and Douglas Stebila (Eds.), Vol. 14927. Springer, Cham, 309–345. https://doi.org/10.1007/978-3-031-68397-8_10

[23] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. 2022. Structure-Aware Private Set Intersection, with Applications to Fuzzy Matching. In *CRYPTO 2022, Part I (LNCS)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.), Vol. 13507. Springer, Cham, 323–352. https://doi.org/10.1007/978-3-031-15802-5_12

[24] Gayathri Garimella, Mike Rosulek, and Jaspal Singh. 2023. Malicious Secure, Structure-Aware Private Set Intersection. In *CRYPTO 2023, Part I (LNCS)*, Helena Handschuh and Anna Lysyanskaya (Eds.), Vol. 14081. Springer, Cham, 577–610. https://doi.org/10.1007/978-3-031-38557-5_19

[25] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. 1999. Similarity Search in High Dimensions via Hashing. In *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann, 518–529. http://www.vldb.org/conf/1999/P49.pdf

[26] Mayank Goswami, Rasmus Pagh, Francesco Silvestri, and Johan Sivertsen. 2017. Distance Sensitive Bloom Filters Without False Negatives. In *28th SODA*, Philip N. Klein (Ed.). ACM-SIAM, 257–269. https://doi.org/10.1137/1.9781611974782.17

[27] Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. 2023. Private Web Search with Tiptoe. In *Proceedings of the 29th Symposium on Operating Systems Principles* (Koblenz, Germany) *(SOSP '23)*. Association for Computing Machinery, New York, NY, USA, 396–416. https://doi.org/10.1145/3600006.3613134

[28] Alexandra Henzinger, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In *USENIX Security 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[29] Laura Hetz, Thomas Schneider, and Christian Weinert. 2023. Scaling Mobile Private Contact Discovery to Billions of Users. In *ESORICS 2023, Part I (LNCS)*, Gene Tsudik, Mauro Conti, Kaitai Liang, and Georgios Smaragdakis (Eds.), Vol. 14344. Springer, Cham, 455–476. https://doi.org/10.1007/978-3-031-50594-2_23

[30] Yu Hua, Bin Xiao, Bharadwaj Veeravalli, and Dan Feng. 2012. Locality-Sensitive Bloom Filter for Approximate Membership Query. *IEEE Trans. Comput.* 61, 6 (June 2012), 817–830. https://doi.org/10.1109/TC.2011.108

[31] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endow.* 9, 1 (Sept. 2015), 1–12. https://doi.org/10.14778/2850469.2850470

[32] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *30th ACM STOC*. ACM Press, 604–613. https://doi.org/10.1145/276698.276876

[33] Daniel Kales, Christian Rechberger, Thomas Schneider, Matthias Senker, and Christian Weinert. 2019. Mobile Private Contact Discovery at Scale. In *USENIX Security 2019*, Nadia Heninger and Patrick Traynor (Eds.). USENIX Association, 1447–1464. https://www.usenix.org/conference/usenixsecurity19/presentation/kales

[34] Adam Kirsch and Michael Mitzenmacher. [n.d.]. *Distance-Sensitive Bloom Filters*. 41–50. https://doi.org/10.1137/1.9781611972863.4 arXiv:https://epubs.siam.org/doi/pdf/10.1137/1.9781611972863.4

[35] Ágnes Kiss, Jian Liu, Thomas Schneider, N. Asokan, and Benny Pinkas. 2017. Private Set Intersection for Unequal Set Sizes with Mobile Applications. *PoPETs* 2017, 4 (Oct. 2017), 177–197. https://doi.org/10.1515/popets-2017-0044

[36] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private Blocklist Lookups with Checklist. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 875–892. https://www.usenix.org/conference/usenixsecurity21/presentation/kogan

[37] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 484–493. https://doi.org/10.14778/1920841.1920904

[38] Kristin Lauter, Sreekanth Kannepalli, Kim Laine, and Radames Cruz Moreno. [n.d.]. *Password Monitor: Safeguarding Passwords in Microsoft Edge*. Microsoft Research Blog. https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/

[39] Rishabh Kumar, Hari Prasanna P, Mukund Rungta, Swetha Kashinath Phuleker, Hemant Tiwari, and Vanraj Vala. 2021. FBF: Bloom Filter for Fuzzy Membership Queries on Strings. In *15th IEEE International Conference on Semantic Computing, ICSC 2021, Laguna Hills, CA, USA, January 27-29, 2021*. IEEE, 25–32. https://doi.org/10.1109/ICSC50631.2021.00010

[40] Arthur Lazzaretti and Charalampos Papamanthou. 2024. Single Pass Client-Preprocessing Private Information Retrieval. In *USENIX Security 2024*, Davide Balzarotti and Wenyuan Xu (Eds.). USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/lazzaretti

[41] Jingyu Li, Zhicong Huang, Min Zhang, Cheng Hong, Jian Liu, Tao Wei, and Wenguang Chen. 2025. Panther: Private Approximate Nearest Neighbor Search in the Single Server Setting. In *ACM CCS 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 365–379. https://doi.org/10.1145/3719027.3765190

[42] Chengyu Lin, Zeyu Liu, Peihan Miao, and Max Tromanhauser. 2025. Finding Balance in Unbalanced PSI: A New Construction from Single-Server PIR. *CiC* 2, 1 (2025), 27. https://doi.org/10.62056/angy4fvtw

[43] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. 2014. SK-LSH: an efficient index structure for approximate nearest neighbor search. *Proc. VLDB Endow.* 7, 9 (May 2014), 745–756. https://doi.org/10.14778/2732939.2732947

[44] Guillaume Marçais, Dan DeBlasio, Prashant Pandey, and Carl Kingsford. 2019. Locality-sensitive hashing for the edit distance. *Bioinformatics* 35, 14 (07 2019), i127–i135. https://doi.org/10.1093/bioinformatics/btz354

[45] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*. Pmlr, 1273–1282.

[46] Shengzhe Meng, Xiaodong Wang, Xv Zhou, and Bei Liang. 2026. Unbalanced Fuzzy Private Set Intersection for L_infinity Distance: Achieving Sublinear Communication with Large Set Size. In *USENIX Security 2026*. USENIX Association.

[47] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized Batch Private Information Retrieval. In *2023 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 437–452. https://doi.org/10.1109/SP46215.2023.10179329

[48] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. 2015. Phasing: Private Set Intersection Using Permutation-based Hashing. In *USENIX Security 2015*, Jaeyeon Jung and Thorsten Holz (Eds.). USENIX Association, 515–530. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas

[49] Lucas Piske, Jaspal Singh, Ni Trieu, Vladimir Kolesnikov, and Vassilis Zikas. 2025. Distance-Aware OT with Application to Fuzzy PSI. In *ACM CCS 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM Press, 4679–4691. https://doi.org/10.1145/3719027.3744857

[50] Stanford Network Analysis Project. 2010. Gowalla Dataset. https://snap.stanford.edu/data/loc-gowalla.html.

[51] Tim Rausch, Sylvain Chatel, and Wouter Lueks. 2026. XDup: Privacy-Preserving Deduplication for Humanitarian Organizations Using Fuzzy PSI. In *2026 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 829–847. https://doi.org/10.1109/SP63933.2026.00140

[52] Sacha Servan-Schreiber, Simon Langowski, and Srinivas Devadas. 2022. Private Approximate Nearest Neighbor Search with Sublinear Communication. In *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 911–929. https://doi.org/10.1109/SP46214.2022.9833702

[53] Harsha Vardhan Simhadri, Martin Aumüller, Amir Ingber, Matthijs Douze, George Williams, Magdalen Manohar, Dmitry Baranchuk, Edo Liberty, and Ravishankar Krishnaswamy. 2022. Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search. In *Proceedings of the NeurIPS 2021 Competitions and Demonstrations Track (Proceedings of Machine Learning Research)*, Vol. 176. 177–189.

[54] A. Z. Tan, H. Yu, L. Cui, and Q. Yang. 2023. Towards Personalized Federated Learning. *IEEE Transactions on Neural Networks and Learning Systems* 34, 12 (2023), 9587–9603. https://doi.org/10.1109/TNNLS.2022.3160699

[55] Ni Trieu, Kareem Shehata, Prateek Saxena, Reza Shokri, and Dawn Song. 2020. Epione: Lightweight Contact Tracing with Strong Privacy. *IEEE Data Eng. Bull.* 43, 2 (2020), 95–107. http://sites.computer.org/debull/A20june/p95.pdf

[56] Juan Ramón Troncoso-Pastoriza, Stefan Katzenbeisser, and Mehmet Celik. 2007. Privacy preserving error resilient dna searching through oblivious automata. In *ACM CCS 2007*, Peng Ning, Sabrina De Capitani di Vimercati, and Paul F. Syverson (Eds.). ACM Press, 519–528. https://doi.org/10.1145/1315245.1315309

[57] Erkam Uzun, Simon P. Chung, Vladimir Kolesnikov, Alexandra Boldyreva, and Wenke Lee. 2021. Fuzzy Labeled Private Set Intersection with Applications to Private Real-Time Biometric Search. In *USENIX Security 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 911–928. https://www.usenix.

org/conference/usenixsecurity21/presentation/uzun
[58] Aron van Baarsen and Sihang Pu. 2024. Fuzzy Private Set Intersection with Large Hyperballs. In *EUROCRYPT 2024, Part V (LNCS)*, Marc Joye and Gregor Leander (Eds.), Vol. 14655. Springer, Cham, 340–369. https://doi.org/10.1007/978-3-031-58740-5_12
[59] Aron van Baarsen and Sihang Pu. 2025. Fuzzy Private Set Intersection from VOLE. In *ASIACRYPT 2025, Part V (LNCS)*, Goichiro Hanaoka and Bo-Yin Yang (Eds.), Vol. 16249. Springer, Singapore, 327–360. https://doi.org/10.1007/978-981-95-5116-3_11
[60] Dinusha Vatsalan, Peter Christen, and Vassilios S. Verykios. 2013. A taxonomy of privacy-preserving record linkage techniques. *Information Systems* 38, 6 (2013), 946–969. https://doi.org/10.1016/j.is.2012.11.005
[61] Jiuqi Wei, Botao Peng, Xiaodong Lee, and Themis Palpanas. 2024. DET-LSH: A Locality-Sensitive Hashing Scheme with Dynamic Encoding Tree for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 9 (May 2024), 2241–2254. https://doi.org/10.14778/3665844.3665854
[62] Ruidi Wei and Florian Kerschbaum. 2023. Cryptographically Secure Private Record Linkage using Locality-Sensitive Hashing. *Proc. VLDB Endow.* 17, 2 (Oct. 2023), 79–91. https://doi.org/10.14778/3626292.3626293
[63] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv preprint arXiv:1708.07747* (2017).
[64] Xinpeng Yang, Meng Hao, Chenkai Weng, Robert H. Deng, Yonggang Wen, and Tianwei Zhang. 2026. Efficient Fuzzy Private Set Intersection from Secret-Shared OPRF. In *2026 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 2442–2461. https://doi.org/10.1109/SP63933.2026.00172
[65] Qingsong Ye, Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. 2010. Efficient Fuzzy Matching and Intersection on Private Datasets. In *ICISC 09 (LNCS)*, Donghoon Lee and Seokhie Hong (Eds.), Vol. 5984. Springer, Berlin, Heidelberg, 211–228. https://doi.org/10.1007/978-3-642-14423-3_15
[66] Foo Yee Yeo and Jason H. M. Ying. 2025. Private Set Intersection and other Set Operations in the Third Party Setting. In *USENIX Security 2025*, Lujo Bauer and Giancarlo Pellegrino (Eds.). USENIX Association, 7723–7742. https://www.usenix.org/conference/usenixsecurity25/presentation/yeo
[67] Bolong Zheng, Xi Zhao, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S. Jensen. 2020. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *Proc. VLDB Endow.* 13, 5 (Jan. 2020), 643–655. https://doi.org/10.14778/3377369.3377374
[68] Mingxun Zhou, Wei-Kai Lin, Yiannis Tselekounis, and Elaine Shi. 2023. Optimal Single-Server Private Information Retrieval. In *EUROCRYPT 2023, Part I (LNCS)*, Carmit Hazay and Martijn Stam (Eds.), Vol. 14004. Springer, Cham, 395–425. https://doi.org/10.1007/978-3-031-30545-0_14
[69] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2024. Piano: Extremely Simple, Single-Server PIR with Sublinear Server Computation. In *2024 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 4296–4314. https://doi.org/10.1109/SP54263.2024.00055
[70] Mingxun Zhou, Elaine Shi, and Giulia Fanti. 2025. Pacmann: Efficient Private Approximate Nearest Neighbor Search. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net. https://openreview.net/forum?id=yQcFniousM

A ADDITIONAL CONTENT ON PRELIMINARIES

A.1 PIR Correctness and Privacy

Below we define the properties of *correctness* and *privacy* for PIR as defined in Def. 5.

Definition 10 (PIR Correctness). *Let Π_{PIR} be a single-server PIR protocol for databases $D \in \mathcal{D}^N$. We say that Π_{PIR} **is correct** if for every database $D \in \mathcal{D}^N$ and every index $\mathsf{idx} \in [N]$,*

$$\Pr\left[\mathbf{Exp}^{\mathsf{corr}}_{\Pi_{\mathsf{PIR}}}(1^\lambda, D, \mathsf{idx}) = 1\right] \geq 1 - \mathsf{negl}(\lambda).$$

Definition 11 (PIR Privacy). *Let Π_{PIR} be a single-server PIR protocol for databases $D \in \mathcal{D}^N$. We say that Π_{PIR} **is private** if for every PPT adversarial server $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$, there exists a PPT simulator $\mathsf{Sim} = (\mathsf{Sim}_0, \mathsf{Sim}_1)$ such that for every $D \in \mathcal{D}^N$ and $\mathbf{qry} \in [N]$,*

$$\mathbf{Real}_{\Pi_{\mathsf{PIR}},\mathcal{A}}(1^\lambda, D, \mathbf{qry}) \approx_c \mathbf{Ideal}_{\Pi_{\mathsf{PIR}},\mathcal{A},\mathsf{Sim}}(1^\lambda, D).$$

$\mathbf{Exp}^{\mathsf{corr}}_{\Pi_{\mathsf{PIR}}}(1^\lambda, D, \mathsf{idx})$

$\mathsf{pp} \leftarrow \mathsf{GenPP}(1^\lambda, |D|)$
$(\mathsf{st}_C; \mathsf{st}_S) \leftarrow \mathsf{Setup}(\mathsf{pp}; (\mathsf{pp}, D))$
$((\mathsf{st}'_C, \mathsf{resp}); \perp) \leftarrow \mathsf{Query}((\mathsf{st}_C, \mathsf{idx}); (\mathsf{st}_S, D))$
return $(\mathsf{resp} = D[\mathsf{idx}])$

$\mathbf{Real}_{\Pi_{\mathsf{PIR}},\mathcal{A}}(1^\lambda, D, \mathbf{qry})$

$\mathsf{pp} \leftarrow \mathsf{GenPP}(1^\lambda, |D|)$
$(\mathsf{st}_C, \mathsf{st}_{\mathcal{A}}) \leftarrow \left\langle \mathrm{C}^{\mathsf{Setup}}(\mathsf{pp}), \mathcal{A}_0(\mathsf{pp}, D) \right\rangle$
$(\mathsf{st}'_C, \mathsf{resp}, \mathsf{st}'_{\mathcal{A}}) \leftarrow \left\langle \mathrm{C}^{\mathsf{Query}}(\mathsf{st}_C, \mathbf{qry}), \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, D) \right\rangle$
$\beta \leftarrow \mathcal{A}_1(\mathsf{st}'_{\mathcal{A}})$
return β

$\mathbf{Ideal}_{\Pi_{\mathsf{PIR}},\mathcal{A},\mathsf{Sim}}(1^\lambda, D)$

$\mathsf{pp} \leftarrow \mathsf{GenPP}(1^\lambda, |D|)$
$(\mathsf{st}_S, \mathsf{st}_{\mathcal{A}}) \leftarrow \left\langle \mathsf{Sim}_0(1^\lambda, \mathsf{pp}, |D|), \mathcal{A}_0(\mathsf{pp}, D) \right\rangle$
$(\mathsf{st}'_S, \mathsf{st}'_{\mathcal{A}}) \leftarrow \left\langle \mathsf{Sim}_1(\mathsf{st}_S), \mathcal{A}_1(\mathsf{st}_{\mathcal{A}}, D) \right\rangle$
$\beta \leftarrow \mathcal{A}_1(\mathsf{st}'_{\mathcal{A}})$
return β

Figure 10: The correctness experiment (top) and the real (middle) and ideal (bottom) experiments for multi-query PIR privacy. Here, $\langle \mathrm{C}, \mathcal{A} \rangle$ denotes a protocol execution between an honest client and adversarial server.

A.2 Euclidean distance LSH (E2LSH)

Datar et al. [15] introduced an LSH family $\mathcal{H}$ for vectors of dimension d under the ℓ_p norm. We focus here on the Euclidean case $p = 2$. A hash function h in this family is defined using a random vector $\mathbf{a} \in \mathbb{R}^d$ and a random offset $b \in [0, w]$, where $w \in \mathbb{R}_{>0}$ is a fixed width parameter. The hash function h parameterized by $\mathbf{a}$ and b is defined as

$$h_{\mathbf{a},b}(\mathbf{x}) = \left\lfloor \frac{\langle \mathbf{a}, \mathbf{x} \rangle + b}{w} \right\rfloor, \tag{1}$$

where $\langle \cdot, \cdot \rangle$ is the standard dot product, i.e., $\langle \mathbf{a}, \mathbf{x} \rangle = \sum_{i=1}^{d} a_i x_i$. The LSH family consists of multiple hash functions with independently sampled $\mathbf{a}$ and b. For any pair $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ with $\|\mathbf{x} - \mathbf{y}\|_2 = c$, the collision probability $\Pr[H(\mathbf{x}) = H(\mathbf{y})]$ is given by

$$\Pr[h_{\mathbf{a},b}(\mathbf{x}) = h_{\mathbf{a},b}(\mathbf{y})] = \int_0^w \frac{1}{c} f_2\left(\frac{t}{c}\right)\left(1 - \frac{t}{w}\right) dt, \tag{2}$$

where $f_2(t)$ is the probability density function of the standard normal distribution.

B ADDITIONAL CONTENT ON THE FUZZY PROBABILISTIC DATA STRUCTURE

B.1 Proof of Theorem 3.1

Proof. Fix any set Y inserted into the Fuzzy PDS, and any query point $q \in \mathcal{U}$. We proceed in two cases.

Case 1: Near case. Suppose there exists some $y^* \in Y$ such that $\mathsf{dist}(q, y^*) \le r_i$. For a fixed $\ell \in [L]$, success requires that

$$H_{\ell,k}(q) = H_{\ell,k}(y^*) \text{ for all } k \in [K].$$

Since $\mathsf{dist}(q, y^*) \le r_i$, by the definition of an LSH family (Def. 2), for each $k \in [K]$,

$$\Pr\big[H_{\ell,k}(q) = H_{\ell,k}(y^*)\big] \ge p_1.$$

By independence of the K hash evaluations, the probability that all $H_{\ell,k}$ collide is $\ge p_1^K$. Moreover, the LSH values are concatenated and subsequently hashed using a collision-resistant hash function H as

$$G_\ell(q) = H(H_{\ell,1}(q), \ldots, H_{\ell,K}(q)).$$

Thus, if $H_{\ell,k}(q) = H_{\ell,k}(y^*)$ for all $k \in [K]$, then, since the outer hash H is deterministic, $G_\ell(q) = G_\ell(y^*)$. Note that any additional collisions introduced by this final deterministic hash can only increase the probability of acceptance, so the above probability gives a valid lower bound on the final hashes colliding.

Now consider the L independent partitions. The probability that none of them collide is $\le (1 - p_1^K)^L$. Thus, with probability at least

$$1 - (1 - p_1^K)^L,$$

there exists $\ell \in [L]$ such that $G_\ell(q) = G_\ell(y^*)$.

Fix such an ℓ. During insertion, the value $G_\ell(y^*)$ was inserted into D. During query, the corresponding value $G_\ell(q)$ is checked against the same PDS value in D. Since these values are equal and the underlying PDS has false-negative probability 0, this lookup returns membership with probability 1. Since the check in the PDS succeeds, the final fuzzy PDS output is $b = 1$.

It follows that

$$\Pr[b = 1 \mid \exists y \in Y : \mathsf{dist}(q, y) \le r_i] \ge 1 - (1 - p_1^K)^L.$$

Thus, we may set

$$\rho_1 \ge 1 - (1 - p_1^K)^L.$$

Case 2: Far case. Now suppose that for all $y \in Y$, $\mathsf{dist}(q, y) \ge r_o$. For a fixed LSH instance $\ell \in [L]$ and a fixed element $y^* \in Y$, define

$$E_\ell := \bigwedge_{k \in [K]} \big(H_{\ell,k}(q) = H_{\ell,k}(y^*)\big).$$

That is, E_ℓ is the event that all K LSH values of q collide with those of y^* in partition ℓ. Since $\mathsf{dist}(q, y^*) \ge r_o$, by the definition of an LSH family (Def. 2), for each $k \in [K]$,

$$\Pr\big[H_{\ell,k}(q) = H_{\ell,k}(y^*)\big] \le p_2.$$

By independence of the K hash evaluations, we thus have $\Pr[E_\ell] \le p_2^K$. The value checked in the PDS, however, is the hash of the concatenation $G_\ell(\cdot)$. For a fixed $y^* \in Y$, the event $G_\ell(q) = G_\ell(y^*)$ can occur either because E_ℓ occurs, or because the outer hash H collides on two distinct LSH tuples. Let $p_H = \mathsf{negl}(\lambda)$ denote an upper bound on the outer hash's collision probability. Then

$$\Pr[G_\ell(q) = G_\ell(y^*)] \le p_2^K + \mathsf{negl}(\lambda).$$

Taking a union bound over all $y \in Y$, the probability that the derived fuzzy value $G_\ell(q)$ matches some value in D is at most

$$\Pr\big[\exists y \in Y : G_\ell(q) = G_\ell(y)\big] \le |Y|(p_2^K + p_H).$$

Thus, for a fixed ℓ, the probability that the lookup succeeds because of a collision with some inserted fuzzy value is at most $|Y|(p_2^K + \mathsf{negl}(\lambda))$. Even if no such collision occurs, the underlying PDS may still return a false positive. Since the false-positive probability of Σ is at most p_{fpp}, the probability that repetition ℓ incorrectly returns membership is at most

$$|Y|(p_2^K + \mathsf{negl}(\lambda)) + p_{\mathsf{fpp}}.$$

Finally, the Fuzzy PDS combines the L LSH instances using OR logic, so the overall output is $b = 1$ if at least one repetition returns membership. Applying a union bound over all L instances gives

$$\Pr[b = 1 \mid \forall y \in Y : \mathsf{dist}(q, y) \ge r_o] \le L\big(|Y|(p_2^K + \mathsf{negl}(\lambda)) + p_{\mathsf{fpp}}\big).$$

Thus, we may set

$$\rho_2 \le L\big(|Y| p_2^K + p_{\mathsf{fpp}}\big) + \mathsf{negl}(\lambda).$$

By combining the two cases, the theorem follows. □

C ADDITIONAL CONTENT ON THE FUZZY PSI PROTOCOL

C.1 A Note on Spatial Hashing

While our protocol leverages LSH, which was developed in the context of plaintext nearest-neighbor search, one could instead use spatial hashing, which provides different correctness and efficiency guarantees. Spatial hashing was used by Garimella et al. [23] in the context of Fuzzy PSI to decrease the asymptotic complexity of checking whether a point $\mathbf{x} \in X$ is within distance δ of a point $\mathbf{y} \in Y$ under the L_∞ norm.

At a high level, the idea is to tile the input space U^d into equal-sized grid cells e.g., d-dimensional hypercubes of side length 2δ. Each cell is assigned a unique label. To avoid false negatives, for each $\mathbf{x} \in X$ the server inserts the labels of all cells that intersect the δ-ball around $\mathbf{x}$. These labels can then be packed into an oblivious key-value store (OKVS). The client, for each $\mathbf{y} \in Y$, computes the label of the cell containing $\mathbf{y}$ and queries the OKVS using standard PSI techniques. If $\|\mathbf{x} - \mathbf{y}\|_\infty \le \delta$, then the cell containing $\mathbf{y}$ necessarily intersects the δ-ball around $\mathbf{x}$, and hence its label was inserted by the server. However, spatial hashing may still produce false positives: a cell that intersects the δ-ball around $\mathbf{x}$ may contain a client point $\mathbf{y}$ whose distance from $\mathbf{x}$ is greater than δ. Such candidates can be filtered by an additional exact distance check.

Since the encoding is exponential in d, this approach works best for low-dimensional spaces.

C.2 Correctness of our Fuzzy PSI Protocol

Theorem C.1 (Correctness). *Let Π_{FPSI} be the protocol in Fig. 4 and let $\lambda \in \mathbb{N}$ be the security parameter. Let Π_{PIR} be a correct PIR scheme and let Π_{OPRF} be a correct batch OPRF scheme, both with correctness error negligible in λ. Let Σ^* be an insertion-only PRF-wrapped Fuzzy PDS that is $(\rho_1, \rho_2, \delta_i, \delta_o)$-correct.*

At the end of executing $\Pi_{\mathsf{FPSI}}(X; Y)$, the server $\mathbf{S}$ outputs $|X|$ and the client $\mathbf{C}$ outputs a set $Z \subseteq X$ such that, for all $\mathbf{x} \in X$:

- *if there exists $\mathbf{y} \in Y$ such that $\mathsf{dist}(\mathbf{x}, \mathbf{y}) \le \delta_i$, then*
$$\Pr[\mathbf{x} \in Z] \ \ge \ \rho_1 - \mathsf{negl}(\lambda),$$
- *if for all $\mathbf{y} \in Y$ we have $\mathsf{dist}(\mathbf{x}, \mathbf{y}) \ge \delta_o$, then*
$$\Pr[\mathbf{x} \in Z] \ \le \ \rho_2 + \mathsf{negl}(\lambda).$$

Proof. Let Good be the event that all OPRF outputs obtained by the client are equal to the corresponding PRF outputs under $\mathsf{k}_{\mathsf{PRF}}$, and that all PIR queries are correct. Since Π_{OPRF} and Π_{PIR} are correct, except with negligible probability, and since the protocol invokes them only polynomially many times, we have $\Pr[\text{Good}] \geq 1 - \mathsf{negl}(\lambda)$.

Fix $\mathbf{x} \in X$, and let $b_{\mathbf{x}}$ be the membership bit computed by the client for $\mathbf{x}$. The client adds $\mathbf{x}$ to Z if and only if $b_{\mathbf{x}} = 1$.

Conditioned on Good, the computation performed by the client on input $\mathbf{x}$ is as follows (lines 12–18 in Fig. 4):

$$\begin{aligned} \mathbf{v} &\leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(\mathbf{x}) \\ (\widetilde{Q}; \perp) &\leftarrow \Pi_{\mathsf{OPRF}}(\mathbf{v}; \mathsf{k}_{\mathsf{PRF}}) \\ (\mathbf{locs}, \mathbf{info}) &\leftarrow \Sigma^*.\mathsf{GetInfo}(\widetilde{Q}) \end{aligned}$$

Then for each $\mathsf{loc} \in \mathbf{locs}$ it runs

$$(\mathsf{st}_C, \mathsf{resp}; \perp) \leftarrow \Pi_{\mathsf{PIR}}.\mathsf{Query}((\mathsf{st}_C, \mathsf{loc}); (\mathsf{st}_S, D))$$

and appends resp to **resps**. Finally, the client checks:

$$b_{\mathbf{x}} \leftarrow \Sigma^*.\mathsf{CheckMemb}(\mathbf{resps}, \mathbf{info}).$$

Conditioned on Good, the OPRF output must satisfy $\widetilde{Q} = \{F_{\mathsf{k}_{\mathsf{PRF}}}(u)\}_{u \in \mathbf{v}}$. Moreover, each PIR response resp is exactly the value that would be obtained by querying location loc in the server's PDS database D. Hence, conditioned on Good, the vector **resps** is exactly the output of

$$\Sigma^*.\mathsf{Query}(D, \mathbf{locs}).$$

Therefore, conditioned on Good, the client's computation of $b_{\mathbf{x}}$ is equivalent to testing membership using the PRF-wrapped Fuzzy PDS, i.e., by executing the following:

$$\begin{aligned} \mathbf{v} &\leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(\mathbf{x}), \\ (\mathbf{locs}, \mathbf{info}) &\leftarrow \Sigma^*.\mathsf{GetInfo}(\{F_{\mathsf{k}_{\mathsf{PRF}}}(u)\}_{u \in \mathbf{v}}), \\ \mathbf{resps} &\leftarrow \Sigma^*.\mathsf{Query}(D, \mathbf{locs}), \\ b_{\mathbf{x}} &\leftarrow \Sigma^*.\mathsf{CheckMemb}(\mathbf{resps}, \mathbf{info}). \end{aligned}$$

Thus the probability that $b_{\mathbf{x}} = 1$, conditioned on Good, is exactly the acceptance probability of the $(\rho_1, \rho_2, \delta_i, \delta_o)$-correct Σ^*.

We now consider the two cases.

Case 1: Assume that there exists $\mathbf{y} \in Y$ such that $\mathsf{dist}(\mathbf{x}, \mathbf{y}) \leq \delta_i$. By the correctness of Σ^*, conditioned on Good we have $\Pr[b_{\mathbf{x}} = 1 \mid \text{Good}] \geq \rho_1$. Therefore,

$$\begin{aligned} \Pr[\mathbf{x} \in Z] &= \Pr[b_{\mathbf{x}} = 1] \\ &\geq \Pr[b_{\mathbf{x}} = 1 \wedge \text{Good}] \\ &= \Pr[\text{Good}] \Pr[b_{\mathbf{x}} = 1 \mid \text{Good}] \\ &\geq (1 - \mathsf{negl}(\lambda))\rho_1 \\ &\geq \rho_1 - \mathsf{negl}(\lambda). \end{aligned}$$

Case 2: Assume that for all $\mathbf{y} \in Y$, $\mathsf{dist}(\mathbf{x}, \mathbf{y}) \geq \delta_o$. By the correctness of Σ^*, conditioned on Good we have that $\Pr[b_{\mathbf{x}} = 1 \mid \text{Good}] \leq \rho_2$. Thus we have that

$$\begin{aligned} \Pr[\mathbf{x} \in Z] &= \Pr[b_{\mathbf{x}} = 1] \\ &\leq \Pr[b_{\mathbf{x}} = 1 \mid \text{Good}] \Pr[\text{Good}] + \Pr[\neg\text{Good}] \\ &\leq \rho_2 + \mathsf{negl}(\lambda). \end{aligned}$$

Since this argument holds for every $\mathbf{x} \in X$, the theorem follows. □

C.3 Proof of Theorem 4.1

Proof. We proceed using a hybrid argument and describe a simulator $\mathsf{Sim}_{\mathsf{S}}$ that simulates the server's view to show that

$$\left(\mathsf{View}_{\mathsf{S}}^{\Pi}(X, Y), \mathsf{Out}_{\mathsf{C}}^{\Pi}(X, Y)\right) \overset{c}{\approx} \left(\mathsf{Sim}_{\mathsf{S}}(1^{\lambda}, Y, |X|), O_{\mathsf{C}}\right).$$

$\mathsf{Sim}_{\mathsf{S}}$ takes as input 1^{λ}, the set Y, and the set size n of the client, and runs the server's protocol honestly, including sampling $\mathsf{k}_{\mathsf{PRF}}$. It simulates the client messages as follows.

(1) **Simulate offline PIR phase.** In the PIR offline phase, it invokes the PIR setup simulator
$$\mathsf{Sim}_0^{\mathsf{PIR}}(1^{\lambda}, \mathsf{pp}, L \cdot m, L \cdot n)$$
to obtain the client and server PIR states st_C and st_S, respectively.

(2) **Simulate the OPRF.** Then, for each $i \in [|X|]$, it uses the server-view simulator for the OPRF scheme to simulate the corresponding batched OPRF execution. In particular, it invokes
$$\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OPRF}}(1^{\lambda}, n_{\mathsf{bSize}}, \mathsf{k}_{\mathsf{PRF}}),$$
where n_{bSize} is the batch size of the OPRF invocation.

(3) **Simulate online PIR phase.** Finally, to simulate the PIR queries, for each $i \in [|X|]$ and each $\ell \in [L]$, it invokes the PIR query simulator
$$\mathsf{Sim}_1^{\mathsf{PIR}}(\mathsf{st}_S),$$
to obtain the updated state st'_S and sets $\mathsf{st}_S \leftarrow \mathsf{st}'_S$.

We now consider the following sequence of hybrids.

Hybrid $\mathbf{Hyb}_0$: The real honest interaction.

Hybrid $\mathbf{Hyb}_1$: Identical to $\mathbf{Hyb}_0$, except instead of running the PIR setup protocol, the simulator $\mathsf{Sim}_{\mathsf{S}}$ invokes the PIR setup simulator $\mathsf{Sim}_0^{\mathsf{PIR}}(1^{\lambda}, \mathsf{pp}, L \cdot m, L \cdot n)$. By security of the PIR scheme (Def. 11), the view of the server in $\mathbf{Hyb}_1$ is identical to that in $\mathbf{Hyb}_0$ and therefore the two hybrids are indistinguishable.

Hybrid $\mathbf{Hyb}_{2,i}$ for $i \in [n]$: Same as $\mathbf{Hyb}_{2,i-1}$ (or in the case of $\mathbf{Hyb}_{2,1}$ it is the same as $\mathbf{Hyb}_1$) with the following change. The i-th OPRF interaction is replaced with the simulated view of $\mathsf{Sim}_{\mathsf{S}}^{\mathsf{OPRF}}$. By OPRF privacy against the server (Def. 3), this is indistinguishable from the previous hybrid.

Hybrid $\mathbf{Hyb}_{3,i,\ell}$ for $(i, \ell) \in [n] \times [L]$. This hybrid is identical to the previous except that the ℓ-th PIR query for the i-th element in X is replaced by messages generated by the PIR query simulator $\mathsf{Sim}_1^{\mathsf{PIR}}$. By PIR privacy (Def. 11), the server's view in $\mathbf{Hyb}_{3,i,\ell}$ is computationally indistinguishable from the previous hybrid.

Hybrid $\mathbf{Hyb}_{3,n,L}$ is exactly the distribution produced by $\mathsf{Sim}_{\mathsf{S}}$. Since $n \in \mathsf{poly}(\lambda)$ there are a polynomial number of hybrids in the sequence. Thus the views of the server in the simulated and in the real execution are computationally indistinguishable and the theorem follows. □

C.4 Proof of Theorem 4.2

Proof. We prove this using a hybrid argument and define a simulator $\mathsf{Sim}_{\mathsf{C}}$ that simulates the client's view. $\mathsf{Sim}_{\mathsf{C}}$ takes as input the client's set X, the fuzzy intersection $Z \subseteq X$, set size $|Y|$, and the match pattern $\mathsf{match} : X \to \{0,1\}^L$. It runs the client's protocol honestly and simulates the server messages as follows.

(1) **Generate public parameters.** The simulator samples a dummy PRF key $\hat{k}$ and runs the setup algorithm honestly to generate public parameters pp^* and initialize the simulated data structure $\widehat{D}$:

$$(\widehat{D}, \mathsf{pp}^*) \leftarrow \Sigma^*.\mathsf{Setup}(\mathsf{sp}^*, F_{\hat{k}}, m).$$

(2) **Sample OPRF outputs.** In order to simulate the OPRF, the simulator maintains a random-function table T. For every $x \in X$, it computes

$$\mathbf{v}_x \leftarrow \Sigma^*.\mathsf{GetFuzzyVals}(x).$$

Then for every $v \in \mathbf{v}_x$, if $\mathsf{T}[v]$ is undefined, samples

$$\mathsf{T}[v] \leftarrow\!\!\$\ \{0,1\}^\lambda.$$

The simulated OPRF outputs for $x \in X$ are defined as

$$\widehat{Q}_x := \{\mathsf{T}[v]\}_{v \in \mathbf{v}_x}.$$

(3) **Program responses consistent with match pattern.** The simulator now programs $\widehat{D}$ to be consistent with the match pattern match. It does so by first computing the query locations and auxiliary information for every $x \in X$ as

$$(\mathbf{locs}_x, \mathbf{info}_x) \leftarrow \Sigma^*.\mathsf{GetInfo}(\widehat{Q}_x)$$

and then inserting them into the data structure as

$$\widehat{D} \leftarrow \Sigma^*.\mathsf{Insert}(\widehat{D}, \mathbf{locs}_x, \mathbf{info}_x)$$

Then for every (x, ℓ) such that $\mathsf{match}[x][\ell] = 0$, the simulator overwrites the corresponding location $\mathsf{loc}_{x,\ell} \in \mathbf{locs}_x[\ell]$ in $\widehat{D}$ with random rejecting values.
Finally, the remaining locations of $\widehat{D}$ are filled with random dummy entries.

(4) **Simulate the OPRF transcripts.** Instead of executing the OPRF protocol, the simulator invokes the client-view OPRF simulator on the client's OPRF inputs and the programmed outputs:

$$\mathsf{Sim}_{\mathsf{C}}^{\mathsf{OPRF}}(1^\lambda, \{\mathbf{v}_x\}_{x \in X}, \{\widehat{Q}_x\}_{x \in X}).$$

(5) **Simulate the PIR interaction.** When the client issues PIR queries for the locations in $\mathbf{locs}_x$, the simulator answers using the simulated database $\widehat{D}$, or invokes the client-view PIR simulator if PIR is modeled ideally. The resulting reconstructed responses are exactly those programmed above, and therefore the client obtains the match pattern match.

We now consider the following sequence of hybrids.

Hybrid $\mathbf{Hyb}_0$: The real honest protocol interaction.

Hybrid $\mathbf{Hyb}_1$: Same as $\mathbf{Hyb}_0$, except instead of using the output of the PRF $F_{\mathsf{k}_{\mathsf{PRF}}}$ evaluated on the client's OPRF inputs, replace them with a random function. By the pseudorandomness property of the PRF, $\mathbf{Hyb}_0$ and $\mathbf{Hyb}_1$ are computationally indistinguishable.

Hybrid $\mathbf{Hyb}_2$. Same as $\mathbf{Hyb}_1$, except replace the real OPRF transcripts with transcripts generated by the client-view OPRF simulator $\mathsf{Sim}_{\mathsf{C}}^{\mathsf{OPRF}}$, using the same client inputs and the same randomly sampled outputs used to replace the PRF evaluations. By OPRF security against a semi-honest client (Def. 3), $\mathbf{Hyb}_1$ and $\mathbf{Hyb}_2$ are computationally indistinguishable.

Hybrid $\mathbf{Hyb}_3$. Same as $\mathbf{Hyb}_2$, except replace the real fuzzy PDS with the programmed simulated database $\widehat{D}$, which agrees with the real execution on the responses specified by match and contains random dummy values elsewhere. Note that the client accesses the fuzzy PDS via the PIR query algorithm. By the security of the PIR scheme $\mathbf{Hyb}_2$ and $\mathbf{Hyb}_3$ are computationally indistinguishable.

Observe that $\mathbf{Hyb}_3$ is exactly the distribution produced by $\mathsf{Sim}_{\mathsf{C}}$. Since there is a constant number of hybrids in the sequence, the views of the client in the simulated and in the real execution are computationally indistinguishable, and the theorem follows. □