Feasibility of Privacy-Preserving Entity Resolution on Confidential Healthcare Datasets Using Homomorphic Encryption

1st Yixiang Yao, 5th Srivatsan Ravi
University of Southern California
Los Angeles, CA, USA
{yixiangy,srivatsr}@usc.edu

2nd Joseph Cecil
Information Sciences Institute
Waltham, MA, USA
jcecil@isi.edu

3rd Praveen Angyan, 4th Neil Bahroos
SC Clinical and Translational Science Institute
Los Angeles, CA, USA
{praveen.angyan,neil.bahroos}@med.usc.edu

***Abstract*—Patient datasets contain confidential information which is protected by laws and regulations such as HIPAA and GDPR. Ensuring comprehensive patient information necessitates privacy-preserving entity resolution (PPER), which identifies identical patient entities across multiple databases from different healthcare organizations while maintaining data privacy. Existing methods often lack cryptographic security or are computationally impractical for real-world datasets. We introduce a PPER pipeline based on AMPPERE, a secure abstract computation model utilizing cryptographic tools like homomorphic encryption. Our tailored approach incorporates extensive parallelization techniques and optimal parameters specifically for patient datasets. Experimental results demonstrate the proposed method's effectiveness in terms of accuracy and efficiency compared to various baselines.**

***Index Terms*—entity resolution, data privacy, homomorphic encryption**

I. Introduction

Patients visit different healthcare facilities to receive primary care, specialized treatments such as obstetrics or dentistry, urgent or emergency services, or to fill prescriptions. However, optimal and safe healthcare often necessitates a unified medical history for each patient. Moreover, healthcare research is frequently multidisciplinary, necessitating the merging of datasets from various research teams. During health crises like pandemics, it is crucial for organizations to share health datasets swiftly and securely. *Entity resolution (ER)* is the task of finding records that refer to the same entity across different data sources [16]. This technique aligns with the purpose of identifying the same patients or research participants to consolidate data about each individual in multidisciplinary and collaborative research.

When applied to patient datasets, ER faces significant challenges due to privacy regulations such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States and the General Data Protection Regulation (GDPR) in the European Union. Additionally, certain populations in healthcare are at particularly high risk if their private information is exposed, such as individuals with mental disabilities or mental health conditions, HIV, or those experiencing homelessness. In healthcare and research, merging datasets by sharing the full dataset is inherently risky. Even without explicit identifiers, medical data containing rare conditions can be sufficient to uniquely identify a patient.

Existing solutions to address these challenges include the use of *privacy-preserving entity resolution (PPER)* techniques, which enable the matching of records across datasets while keeping the data encoded and ensuring that individual privacy is maintained [20, 10]. AMPPERE [22], as a platform-agnostic PPER framework leveraging either secure multi-party computation or fully homomorphic encryption, is proposed to guarantee the cryptographic security of the data without sacrificing utility. However, as a universal abstract model, AMPPERE encounters efficiency challenges when scaling to large practical datasets, as it requires compatibility with heterogeneous underlying secure multi-party computation protocols and homomorphic encryption schemes.

In this work, we focus on resolving the real-world PPER challenge of healthcare datasets, aiming to ensure HIPAA compliance and minimize the risk of exposing private information. We develop our open-source solution using AMPPERE as the foundation, ensuring precise computational accuracy and provable privacy. We extensively parallelize the execution by employing various techniques and optimize the parameters specifically for the datasets. Subsequently, we conduct comprehensive experiments comparing our optimized method with baselines, as well as ablation studies to verify the effectiveness of each optimized component.

II. Preliminaries and Related Work

A. Definitions

A formal problem of **privacy-preserving entity resolution** (PPER) can be framed as a triple $T = (D, M, E)$, where $D = \{D_1 \dots D_n\}$ represents a collection of n distinct datasets comprising records r, each owned by a different data owner $P = \{P_1 \dots P_n\}$. The encoding or encryption algorithm E is responsible for maintaining the confidentiality of records from each dataset, such that r from each D is transformed into encoded or ciphertext form, denoted as $[\![r]\!] \in E(D)$, where $[\![r]\!]$ explicitly indicates that r is in a privacy-preserving representation (i.e., ciphertext in our setting). The match set

M contains pairs of records that match between any two datasets among the n parties, that is, $M = \{([\![r_i]\!], [\![r_j]\!]) \,|\, r_i = r_j; [\![r_i]\!] \in E(D_k), [\![r_j]\!] \in E(D_m)\}$, where $r_i = r_j$ indicates that r_i and r_j refer to the same entity in the real world.

Conducting PPER naively requires so-called **full comparison** $T = \{(r_i, r_j) \,|\, r_i \in D_k, r_j \in D_m\}$ for all the possible pairs between parties, resulting in a total number of comparisons equivalent to the Cartesian product of the sizes of D_k and D_m (i.e. $|T| = |D_k| \times |D_m|$). Since PPER can be computationally heavy, adopting such a quadratic method is impractical. To eliminate unnecessary comparisons, the **blocking** algorithm is employed to prune T to T' by removing pairs that are unlikely to be part of the solution M. Typically, each record r is represented by a set of **blocking keys** $\beta(r) = (k_1, k_2, \cdots)$, where $\beta(\cdot)$ is the key generation function. Consequently, the **candidate pairs** are the pairs that share at least one blocking key in common, that is, $T' = \{(r_i, r_j) \,|\, |\beta(r_i) \cap \beta(r_j)| > 0, r_i \in D_k, r_j \in D_m\}$.

B. Homomorphic Encryption and CKKS

Homomorphic encryption (HE) enables computation directly on encrypted data, preserving the functional relationship between plaintext and ciphertext [8, 1]. Formally, for plaintext and ciphertext operators $\odot_{\mathcal{M}}$ and $\odot_{\mathcal{C}}$, an encryption function E is homomorphic if $\forall m_1, m_2 \in \mathcal{M}$, $E(m_1 \odot_{\mathcal{M}} m_2) \leftarrow E(m_1) \odot_{\mathcal{C}} E(m_2)$, meaning no decryption occurs during computation. *CKKS (Cheon-Kim-Kim-Song)* [11] is an HE scheme supporting approximate arithmetic on encrypted real or complex numbers, with efficient ciphertext packing. CKKS preserves algorithmic accuracy while ensuring data confidentiality and provides semantic and IND-CPA security [18, 2]. Our AMPPERE-based pipeline adopts CKKS to protect data privacy so that neither data owners nor third parties can infer others' data or computation results.

C. Related Works

PPER Initial research has explored using secure one-way hashing and masking algorithms to protect record content [6, 17], but these fail to handle similar record variants. Some algorithms use probabilistic data structures, such as Bloom filters [15], but these are vulnerable to frequency attacks and require a balance between data privacy and utility. Other approaches modify string matching algorithms [21, 23] yet their applicability remains restricted to specific algorithms within particular application contexts. With advancements in embedding techniques, some methods [13] transform original records into multi-dimensional embedding vectors that preserve key semantics for similarity comparison and clustering. However, these embeddings are not secure and are susceptible to inversion attacks [19]. Recently, methods utilizing secure multi-party computation or homomorphic encryption-based approaches have been explored [14, 9], offering strong privacy guarantees but are still computationally and resource-intensive.

PPER in healthcare Healthcare adoption of PPER remains limited due to challenges in scalability and interpretability. Datavant [1], used by N3C, offers a commercial PPER system [3, 12] that matches encrypted tokens derived from SHA-256 and AES-128. Although identifiers are irrecoverable, token-based matching only supports exact matches, and identical tokens across datasets reduce semantic security.

TABLE I: Basic statistics of patient mortality datasets.

Dataset	# records	SSN %	DOB %	FN %	LN %
D_1	19,274	80.2	100	100	99.99
D_2	859,725	95.3	99.95	99.99	99.99

III. Patient Mortality Datasets

There are numerous PPER tasks in the healthcare domain. In this study, we focus on one specific task related to patient mortality. Patients are recorded as deceased only if they die during care, and Electronic Medical Records (EMR) systems seldom update mortality status afterward. To obtain accurate data, records must be matched with external sources. Our goal is to build an efficient and secure PPER pipeline that integrates external mortality data with EMRs to provide complete and reliable mortality information for research.

Datasets The patient mortality datasets consist of two sub-datasets. Dataset D_1 contains cancer patients with deceased information and has the identifiers shown in Figure 1. Dataset D_2 contains patients from the EMR and has all of the identifiers as D_1. The basic statistics of these datasets are summarized in Table I [2].

Ground truth Ground truth is essential for tuning and evaluating PPER algorithms. It consists of record ID pairs across datasets, $G = \{(i, j) \,|\, r_i = r_j, r_i \in X(D_1), r_j \in X(D_2)\}$, where $X(\cdot)$ is a sampling method. Creating this set often requires manual annotation, but in mortality datasets, unique identifiers simplify the process. MRNs serve as unique identifiers for patients but within an institution, while SSNs provide national uniqueness for linking records. Note we do not use SSN directly for PPER because it is not generally available in other healthcare datasets.

Pre-processing We normalize SSN and DOB formats to standard forms: SSN as `XXX-XX-XXXX` and DOB as zero-padded `MM/DD/YYYY`. We additionally remove invalid SSNs and perform deduplication on each dataset individually, reducing the size of D_1 to 19,274 and D_2 to 859,714.

IV. Domain-Specific Optimizations

A. AMPPERE

AMPPERE is a platform-agnostic, cryptographically secure PPER framework that defines a layered pipeline using primitive operations common to secure multi-party computation and homomorphic encryption schemes. Here, we first briefly introduce how AMPPERE works. Each data owner P_1 and P_2 encrypts record tokens and generates blocks. Encrypted data are sent to P_3 for block merging and deduplication

[1] https://www.datavant.com/

[2] Both datasets contain additional attributes, but only the identifiers share common information that are useful to match entities.

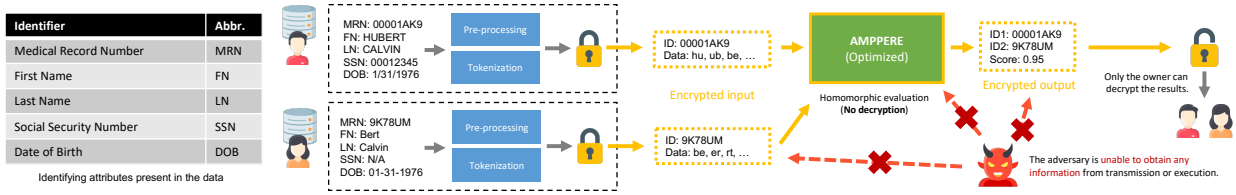

Fig. 1: The overview of the dataset and the pipeline. Arrows and boxes in orange indicate the data is in cipher. Each record, after pre-processing and tokenization, is encrypted into ciphertext. All records from both datasets are then sent to the optimized AMPPERE pipeline, which homomorphically evaluates the potential matches and produces record pairs with scores in encrypted form. Finally, the data owners decrypt the results collectively, whereas the adversary cannot gather any information.

through *Private Matrix Operations*. Due to the limit of the supported operations in security primitives, an interactive operation is required for generating final candidate pairs from the merged block using obfuscation. After that, the entities are resolved using a computationally efficient *Private Set Intersection (PSI)* [4] method called *Vector Rotation*. Final results are collaboratively decrypted by P_1 and P_2.

AMPPERE's abstraction limits scalability on large datasets. We therefore, instead of supporting all compatible security primitives, implement it using the CKKS scheme for its efficient approximate arithmetic and non-interactive flexibility, with OpenFHE [3] as the backend. In the subsequent sections, we describe optimizations for medical datasets, focusing on parallelization at the architecture (chunking), algorithm (SIMD), and programming (multi-processing) levels, and non-interactive approximation operators for improved efficiency.

B. Chunking

Chunking is widely used in ER to divide large datasets into smaller, manageable units for efficient processing. Typically, chunking is based on the output of blocking [7]. As depicted in the left segment of Figure 2, after the autonomous generation of blocks in P_1 and P_2, they need to be shared so that these *single-side blocks* are merged and deduplicated according to their blocking keys (b_k are blocking keys and r_{ij}s are associated records). After that, and these pairs are chunked and distributed to computation units (C_1, C_2, C_3).

However, blocking-based chunking is inefficient for encrypted data, as serialization and data transfer add heavy overhead. Because record-to-block assignments are unpredictable, each computation unit must access all encrypted records, leading to arbitrary distribution and poor I/O efficiency.

To address this, we elevate record pairs to be first-class citizens and apply chunking directly on them. As illustrated in the right segment of the Figure 2, P_1 and P_2 send their chunked encrypted records and corresponding single-side blocks to computation units C_is, where private merging and deduplication are performed using AMPPERE's *Private Matrix Operation*. Each matrix entry (i, j) indicates whether r_i and r_j share a block. Consequently, since the distribution of encrypted records chunks on C_is are not arbitrary and the block merging and deduplication are via the private matrix

[3]https://openfhe.org

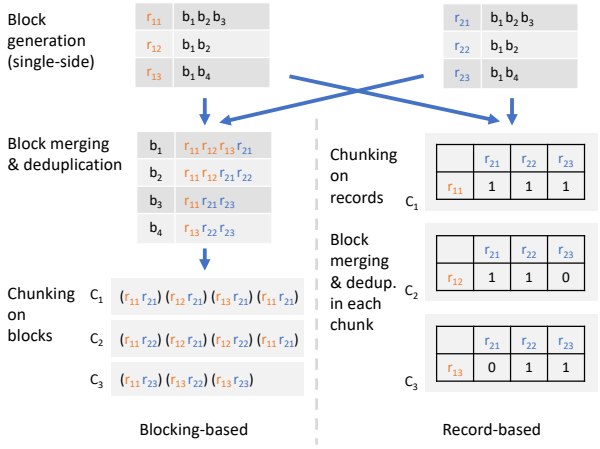

Fig. 2: Chunking strategy.

operation, the record-based chunking minimizes I/O overhead, and leaks no information.

C. SIMD

SIMD stands for Single Instruction Multiple Data, in which a single instruction is applied to multiple data elements at the same time, rather than processing each data element individually. This approach is particularly efficient for tasks in homomorphic encryption evaluation that involve performing the same operation on a large set of data. For example, adding numerous integer pairs sequentially in a loop can be achieved through a single addition operation between two vectors, where the original integer pairs are packed into them respectively. While SIMD greatly improves parallel computation efficiency, integrating it into HE program is challenging. Not all operators support parallel execution, data must be pre-batched and aligned, and some operations, such as obtaining the vector length, become inaccessible once elements are encrypted in batch.

In AMPPERE, each encrypted record is represented as a list of encrypted tokens, $[\![r]\!] = [[\![t_1]\!], \cdots, [\![t_n]\!]]$. To leverage the benefits of SIMD, encryption is applied directly to the entire list, $[\![r]\!] = [\![[t_1, \cdots, t_n]]\!]$. Correspondingly, this modification of the data representation leads to the change of several core operations in the original AMPPERE, therefore each of them must be restructured to support SIMD. For instance, `EEq` performs element-wise equality checks. With SIMD, the

difference between two records can be computed in a single operation rather than iterating over each pair in a loop.

D. Parallel Programming

Modern computers are equipped with multiple CPU cores, and multi-processing/multi-threading are the techniques in which multiple processes/threads are executed concurrently by utilizing multiple CPU cores. We adopt such techniques as programming-level parallelization to utilize the available computation resources fully.

In some simple circumstances, loops can run in parallel. Concretely, decryption is performed in parallel to retrieve the obfuscated pairs, while converting the private candidate pair matrix (for block merging and deduplication) into a candidate pair list is also executed concurrently. In more sophisticated scenarios, we combine parallel programming with SIMD (Section IV-C). For example, AMPPERE employs a private matrix operation for block merging and deduplication, where the encrypted boolean value of an entry at position $(\llbracket i \rrbracket, \llbracket j \rrbracket)$ in the matrix indicates whether $\llbracket r_i \rrbracket$ and $\llbracket r_j \rrbracket$ belong to the same block. Consequently, an $m \times n$ private matrix contains $m \cdot n$ encrypted elements. In our optimization, we convert the matrix representation to m encrypted n-dimensional vectors. This representation enables each n-dimensional vector to utilize SIMD, while the m vectors can be executed concurrently across multiple threads for a certain operation.

E. Non-Interactive Comparison

Logical comparisons are fundamental to decision-making in algorithms and control flow structures like loops and conditional statements in programming. AMPPERE utilizes a crucial logical operator $\texttt{EEq}(a, b)$ for comparing record IDs. However, for pure arithmetic homomorphic encryption schemes, such as CKKS, they do not support any logical operators. In AMPPERE, `EEq` is implemented with an inverse operation and a process of random numbers and interaction between parties.

In recent years, a new solution, solely relying on addition and multiplication, have been proposed for computing inverse [5]. It has two functions, `Inv` for inverse and `Comp` for comparison (based on `Inv`). The $\texttt{Comp}(a, b)$ returns 0, 0.5, and 1 for the cases that $a < b$, $a = b$, and $a > b$, respectively. To use `Comp`, inputs a and b are required to be in a certain fractional range. Specifically, we transform records' globally unique IDs to in-chunk IDs, and ensuring the chunk size does not exceed the preset batch size in CKKS, that is, $i \in \{0, 1, \ldots, batchSize - 1\}$. We then rescale i as $\hat{i} = \frac{1}{2} + \frac{i}{batchSize}$, such that $\hat{i} \in \left[\frac{1}{2}, \frac{3}{2}\right)$. Additionally, to build $\texttt{EEq}(a, b)$, the function needs one more step to map the result of $\texttt{Comp}(a, b)$ to Boolean, i.e., $0/1 \mapsto 0$ ($a < b$ / $a > b$), and $0.5 \mapsto 1$ ($a = b$). This can be achieved arithmetically by $\texttt{EEq}(a, b) = 4\,\texttt{Comp}(a, b)\,\texttt{Comp}(b, a) = 4\,\texttt{Comp}(a, b)(1 - \texttt{Comp}(a, b))$.

V. Experiments

A. Settings

Baselines We compare our method with three baselines.

TABLE II: Blocking performance for the ER pipeline.

Pair Completeness (PC)	Reduction Ratio (RR)	$F\text{-}score$
100%	99.996%	99.998%

- **Naive HE adaptation** This is the most basic version of the PPER pipeline using HE. It simply wraps the ER operation with HE (without blocking), that is, *Private Set Intersection (PSI)* [4]. Therefore, such a method finds pairs using a full quadratic comparison over T.
- **Cleartext ER** This is a plaintext version of the full ER pipeline. This operates almost in the same way as the privacy-preserving pipeline: It first computes candidate pairs T' with blocking, then computes similarity scores using ER within T'.
- **AMPPERE** The full PPER pipeline without any domain-specific optimizations.

Metrics We compute *true positive rate* (also called *recall*), *precision*, and *false positive rate* for measuring the ER accuracy. As for the performance of blocking, we use *pairs completeness* ($PC = \frac{M \cap T'}{M}$) to measure the percentage of true pairs that are blocked, and *reduction ratio* ($RR = 1 - \frac{|T'|}{|T|}$) to measure how well the method reduces the number of candidate pairs. F-1 is employed to harmonically demonstrate the overall performance of blocking.

Environment The experiments are conducted on a privacy-protected Linux instance with 14G RAM and a 4-core CPU. The pipeline implementation uses OpenFHE version 1.1.2.

B. Main Results

We report results from two perspectives: blocking and ER performance, and runtime.

The effectiveness of blocking is represented in Table II. The pair completeness (PC) is 100%, indicating that the blocking algorithm captures all potential pairs. The reduction ratio RR is 99.996%, meaning the candidate pair set T' is only 0.004% of the full comparison set T. Therefore, applying blocking to this dataset preserves all potential pairs while significantly reducing the pairwise comparisons by more than 99%. As for the ER performance, we access it with different thresholds t for the record-matching scores. If the score is greater than t, the record pair is identified as the same entity. As shown in Figure 3, the pipeline achieves a high recall (over 92%) for the highest threshold, and slightly lower thresholds decrease the precision only negligibly. Moreover, a threshold of around 0.5 achieves nearly perfect recall at approximately 90% precision and a false positive rate of less than 0.0001%. The evaluation above is performed on the cleartext baseline. Both AMPPERE and the optimized pipeline exhibit identical performance because our implementation does not sacrifice any matching performance.

For runtime, with chunk size 50, our optimized pipeline is 24× faster than AMPPERE and 447× faster than a naive HE baseline (Figure 4). AMPPERE itself is 19× faster than the naive HE approach. With more resources, e.g., 96 CPU cores, our solution could achieve further speedup.

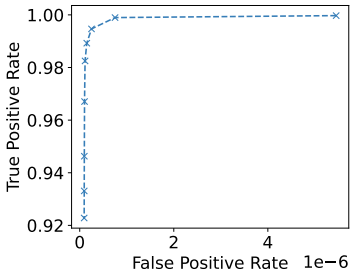

Fig. 3: ROC curve for the ER system.

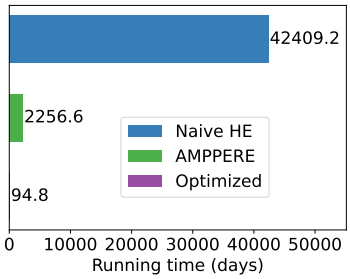

Fig. 4: Estimated runtime for optimized system v.s. two baselines.

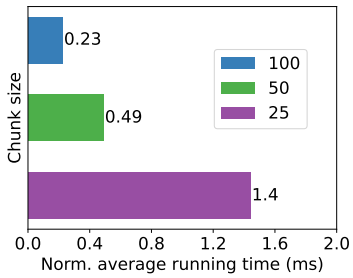

Fig. 5: Normalized average time for different chunk sizes.

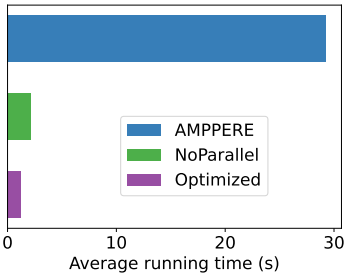

Fig. 6: Time comparison for ablation experiments.

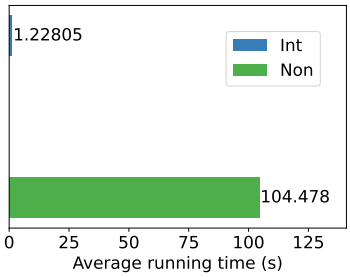

Fig. 7: Time comparison for interactive vs. non-interactive pipeline.

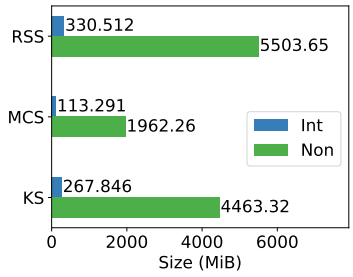

Fig. 8: Space comparison for interactive vs. non-interactive pipeline.

C. Ablation Studies for Pipeline Optimizations

Chunking Since the encrypted dataset cannot fit into memory, we partition it into chunks of size 25, 50, and 100. Because the optimized method employs record-based chunking, the potential number of record pairs for a chunk pair equals the square of the original chunk size: $25^2 = 625$, $50^2 = 2,500$, and $100^2 = 10,000$. Consequently, we normalize the running time by the squared chunk size. We present the results in Figure 5. The results show that doubling the chunk size greatly lowers the normalized running time, indicating that the relationship between chunk size and normalized time is non-linear. The poorer performance observed with smaller chunks may be attributed to more frequent memory swapping, disk I/O, and serialization/deserialization of the cipher data. Practically, batch size should be set as large as memory allows for optimal speed.

Parallel Programming & SIMD In this experiment, we assess the impact of parallel programming and SIMD. We conduct two versions for parallel programming: one with parallel programming enabled and one without. For SIMD, due to its changes to the cipher data representation and corresponding functions, it cannot be isolated in the experiments. Consequently, we use AMPPERE as the baseline and create two variants of the optimized pipeline: SIMD-only (NoParallel) and SIMD combined with Parallel Programming (Optimized). The results are presented in Figure 6. SIMD-only is significantly faster than AMPPERE, highlighting the benefits of processing a batch of data simultaneously. Furthermore, incorporating parallel programming alongside SIMD in the optimized version further reduces execution time by over 40% relative to the SIMD-only version.

Non-Interactive Comparison We first compare the time cost of the interactive and non-interactive pipelines. The non-interactive pipeline (Figure 7), although devoid of interaction and pre-generated random ciphers, operates 85 times slower than the interactive pipeline. This discrepancy arises from the relatively high multiplicative depth and the overhead associated with bootstrapping. Furthermore, we evaluate memory and storage utilization between the interactive and non-interactive pipelines. Specifically, we compare peak memory usage (RSS), average storage usage for each chunk (MCS), and key storage utilized for keys (KS). As demonstrated in Figure 8, the non-interactive pipeline consumes 16.6, 17.3, and 16.6 times more than the corresponding interactive pipeline in each category, respectively. These findings underscore that the non-interactive pipeline necessitates significantly greater memory and storage per chunk.

VI. Conclusion

We applied the CKKS variant within AMPPERE to enable PPER on real-world healthcare data. CKKS ensures HIPAA-compliant data privacy, while our optimizations to data representation and operator execution significantly improve efficiency without reducing accuracy or privacy for large-scale applications.

Acknowledgments

This work was supported in part by NIH award R01AG081571.

References

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. A survey on homomorphic encryption schemes: Theory and implementation. *ACM Computing Surveys (Csur)*, 51(4):1–35, 2018.

[2] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption. In *Proceedings 38th Annual Symposium on Foundations of Computer Science*, pages 394–403. IEEE, 1997.

[3] Elmer V Bernstam, Reuben Joseph Applegate, Alvin Yu, Deepa Chaudhari, Tian Liu, Alex Coda, and Jonah Leshin. Real-world matching performance of deidentified record-linking tokens. *Applied Clinical Informatics*, 13(04):865–873, 2022.

[4] Hao Chen, Kim Laine, and Peter Rindal. Fast private set intersection from homomorphic encryption. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1243–1255, New York, NY, USA, 2017. ACM.

[5] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International conference on the theory and application of cryptology and information security*, pages 415–445. Springer, 2019.

[6] L Dusserre, C Quantin, and H Bouzelat. A one way public key cryptosystem for the linkage of nominal files in epidemiological studies. *Medinfo. MEDINFO*, 1995.

[7] Vasilis Efthymiou, George Papadakis, George Papastefanatos, Kostas Stefanidis, and Themis Palpanas. Parallel meta-blocking for scaling entity resolution over big heterogeneous data. *Information Systems*, 65:137–157, 2017.

[8] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:1–10, 2007.

[9] Tanmay Ghai, Yixiang Yao, and Srivatsan Ravi. Lessons learned: Building a privacy-preserving entity resolution adaptation of ppjoin using end-to-end homomorphic encryption. In *2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 117–124. IEEE, 2023.

[10] Aris Gkoulalas-Divanis, Dinusha Vatsalan, Dimitrios Karapiperis, and Murat Kantarcioglu. Modern privacy-preserving record linkage techniques: An overview. *IEEE Transactions on Information Forensics and Security*, 16:4966–4987, 2021.

[11] Yongsoo Song Jung Hee Cheon, Andrey Kim & Miran Kim. Homomorphic encryption for arithmetic of approximate numbers. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham, 2017. Springer International Publishing.

[12] Daniel Kiernan, Thomas Carton, Sengwee Toh, Jasmin Phua, Maryan Zirkle, Darcy Louzao, Kevin Haynes, Mark Weiner, Francisco Angulo, Charles Bailey, et al. Establishing a framework for privacy-preserving record linkage among electronic health record and administrative claims databases within pcornet®, the national patient-centered clinical research network. *BMC Research Notes*, 15(1):337, 2022.

[13] Bing Li, Yukai Miao, Yaoshu Wang, Yifang Sun, and Wei Wang. Improving the efficiency and effectiveness for bert-based entity resolution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 13226–13233, 2021.

[14] Yehida Lindell. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*, pages 1005–1009. IGI global, 2005.

[15] Frank Niedermeyer, Simone Steinmetzer, Martin Kroll, and Rainer Schnell. Cryptanalysis of basic bloom filters used for privacy preserving record linkage. *German Record Linkage Center, Working Paper Series, No. WP-GRLC-2014-04*, 2014.

[16] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and filtering techniques for entity resolution: A survey. *ACM Computing Surveys (CSUR)*, 53(2):1–42, 2020.

[17] Catherine Quantin, Hocine Bouzelat, FAA Allaert, Anne-Marie Benhamiche, Jean Faivre, and Liliane Dusserre. How to ensure data security of an epidemiological follow-up: quality assessment of an anonymous record linkage procedure. *International journal of medical informatics*, 49(1):117–122, 1998.

[18] Kazue Sako. Semantic security. Encyclopedia of Cryptography and Security (2nd Ed.), 2011.

[19] Congzheng Song and Ananth Raghunathan. Information leakage in embedding models. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*, pages 377–390, 2020.

[20] Dinusha Vatsalan, Ziad Sehili, Peter Christen, and Erhard Rahm. Privacy-preserving record linkage for big data: Current approaches and research challenges. In *Handbook of Big Data Technologies*. Springer, 2017.

[21] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 492–503, 2015.

[22] Yixiang Yao, Tanmay Ghai, Srivatsan Ravi, and Pedro Szekely. Amppere: A universal abstract machine for privacy-preserving entity resolution evaluation. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pages 2394–2403, 2021.

[23] Ruiyu Zhu and Yan Huang. Efficient privacy-preserving general edit distance and beyond. *IACR Cryptology ePrint Archive*, 2017:683, 2017.