Engorgio: An Arbitrary-Precision Unbounded-Size Hybrid Encrypted Database via Quantized Fully Homomorphic Encryption

Song Bian[1], Haowen Pan[1], Jiaqi Hu[1], Zhou Zhang[1], Yunhao Fu[1], Jiafeng Hua[2], Yi Chen[3], Bo Zhang[3], Yier Jin[4], Jin Dong[3]*, and Zhenyu Guan[1]*

[1]Beihang University, [2]Huawei Technology
[3]Beijing Academy of Blockchain and Edge Computing
[4]University of Science and Technology of China

Abstract

This work proposes an encrypted hybrid database framework that combines vectorized data search and relational data query over quantized fully homomorphic encryption (FHE). We observe that, due to the lack of efficient encrypted data ordering capabilities, most existing encrypted database (EDB) frameworks do not support hybrid queries involving both vectorized and relational data. To further enrich query expressiveness while retaining evaluation efficiency, we propose Engorgio, a hybrid EDB framework based on quantized data ordering techniques over FHE. Specifically, we design a new quantized data encoding scheme along with a set of novel comparison and permutation algorithms to accurately generate and apply orders between large-precision data items. Furthermore, we optimize specific query types, including full table scan, batched query, and Top-k query to enhance the practical performance of the proposed framework. In the experiment, we show that, compared to the state-of-the-art EDB frameworks, Engorgio is up to 28×–854× faster in homomorphic comparison, 65×–687× faster in homomorphic sorting and 15×–1,640× faster over a variety of end-to-end relational, vectorized, and hybrid SQL benchmarks. Using Engorgio, the amortized runtime for executing a relational and hybrid query on a 48-core processor is under 3 and 75 seconds, respectively, over a 10K-row hybrid database.

1 Introduction

Hybrid databases (DBs) supporting complex queries that combine high-dimensional vector searching and relational data filtering emerge as a powerful tool in establishing advanced big data applications such as recommendation systems [83], fraud detection [82], retrieval-augmented generation (RAG) systems [47, 60], and many more [89, 91, 94]. In such applications, it is often the case that the data owner and the storage provider are of different geological locations, where the DBs are stored and queried in an outsourced form [97]. While data outsourcing addresses the demands of low-cost and flexible data management [4, 5, 30, 68, 85, 96], images and high-dimensional data stored in the hybrid DBs can be highly sensitive, causing substantial security concerns over data breaching [18, 84].

Facing the security challenges against data outsourcing, we see recent literature that explore the use of fully homomorphic encryption (FHE) in constructing encrypted data management systems. Notably, FHE are utilized to establish encrypted databases (EDBs) [10, 43, 55, 77, 95] and encrypted vector searching (EVS) systems [32, 48, 79, 97, 99]. For example, [10] and [95] propose multi-scheme FHE infrastructures capable of implementing a wide range of common SQL query statements to establish end-to-end EDBs. Similarly, existing works also propose designs of EVS functionalities based FHE, such as encrypted sorting [19, 48] and encrypted k-nearest neighbors [21, 32, 81, 99].

Motivation: Despite the substantial progresses, the current designs of EDB and EVS face two key challenges when handling queries over encrypted data with both relational and vectorized properties: data precision and execution efficiency. First, data items in hybrid DB can vary significantly in formats, ranging from multi-precision integers to gigabyte files, and losing even a single bit in data precision can result in completely incorrect system response. For example, due to insufficient data precision, some existing methods [32, 95] can only sort around 500 to 1,000 data items. Second, the number of data items can be extremely large in DB systems, easily exceeding millions of records. Although some constructions [10, 48, 93] do support various hybrid SQL statements over relatively large number of items, the execution time of such queries can be prohibitive over large-size DBs. For instance, it takes HE^{3}DB [10] nearly 2 days to complete an `ORDER BY` query over 4,096 data items. As a result, the main motivation for this work is to develop a unified hybrid EDB framework that executes SQL queries over arbitrary-precision data with unbounded DB size.

*Corresponding author.

Table 1: Qualitative Comparison of Existing FHE-Based Data Management Systems

	VPIR [69]	Kortekaas [55]	Antonio *et al.* [42]	HEDA [77]	HE^{3}DB [10]	ArcEDB [95]	TFHE-rs [93]	PEGASUS [64]	Zuber *et al.* [99]	Hong *et al.* [48]	Cong *et al.* [32]	Ours
Data Types	Scalar	Scalar	Scalar	Scalar	Scalar	Scalar	Scalar	Hybrid	Vector	Vector	Vector	Hybrid
Sorting	○	○	○	○	●	◐*	●	●	●	●	●	●
Column Sync.	○	○	○	○	●	●	◐+	◐+	◐+	◐+	◐+	●
Top-k	○	○	○	○	○	○	○	●	●	●	●	●
Query Precision†	Medium	Arbitrary	Arbitrary	Low	Medium	Arbitrary	Medium	Medium	Low	Medium	Low	Arbitrary
Ordering Precision†	○	○	○	○	Low	Low	Medium	Low	Low	Medium	Low	Arbitrary
Unbounded-Size	○	○	○	○	○	○	○	○	○	○	○	●
Conversion Free	●	●	○	○	○	○	●	○	●	●	○	●

* Do not support sorting sequences with duplicated elements. + The computational complexity of synchronization is equivalent to sorting.
† Low precision is less than 16 bits; Medium precision is less than 32 bits, or can be improved with the larger parameters, which lead to a decrease in efficiency.

1.1 Our Contribution

To tackle the challenge, in this work, we propose Engorgio, an encrypted database framework tailored for processing hybrid queries over data at scale. We observe that most existing FHE-based encrypted data processing platforms lack the native support for hybrid queries, primarily due to the difficulty in designing highly usable data ordering algorithms. To address such issues, we propose a new set of single-instruction-multi-data (SIMD) FHE operators over quantized ciphertexts to effectively solve the trilemma between query efficiency, data accuracy and operator expressiveness. The key design principle here is to separate the processes of order generation (i.e., comparisons between data items) and order application (i.e., permuting data items), such that an expressive set hybrid SQL statements can be efficiently supported. The main contributions of this work are summarized as follows.

- **A New FHE Infrastructure for Hybrid EDB**: To the best of our knowledge, Engorgio is the first FHE-based EDB framework that is capable of efficiently executing hybrid queries over encrypted data. We propose a quantized FHE scheme equipped with new data ordering operators to accurately handle hybrid SQL queries over large-volume DBs.

- **Homomorphic SIMD Data Ordering**: Based on the quantized FHE scheme, we introduce a new homomorphic comparison operator that can compare arbitrary-precision data in a SIMD fashion. Furthermore, we propose a symmetric sparsity encoding technique to speedup the homomorphic permutation process according to the comparison results, where the permutation complexity can be amortized to $O(1)$.

- **Optimizing Hybrid Query Toolkit**: We develop and optimize a comprehensive list of SQL statements to support various types of hybrid queries over Engorgio. We support most relational statements in the TPC-H benchmark (e.g., `GROUP BY`, `SELECT`, `JOIN`), as well as advanced vectorized statements (e.g., `Distance`, `ORDER BY`, `Limit k`). We also propose a separate design of the sorting (i.e., `HomSort`) and the synchronizing (i.e., `HomSync`) operators, which is crucial for fast evaluation of hybrid queries.

- **Thorough Experiments**: We show that Engorgio achieves 28×–854× faster homomorphic comparison, 65×–687× faster homomorphic sorting, 42×–4,595× faster homomorphic synchronization, and 4×–58× faster homomorphic Top-k over the state-of-the-art (SOTA) solutions. For end-to-end query evaluation, Engorgio runs 15×–621× faster on relational queries, 30×–388× faster on vectorized queries, and 68×–1,640× faster on hybrid queries when comparing to the SOTA FHE-based EDB frameworks.

1.2 Related Works

1.2.1 Encrypted Databases

The study of EDB attracts research interests from across the fields, where numerous protocols and systems are proposed to effectively perform SQL query execution on encrypted data [9, 10, 33, 37, 38, 41, 43, 43, 51, 55, 62, 71–75, 77, 86, 88, 90, 95]. Here, we classify existing EDB works into two main categories: secure multi-party EDB and outsourced EDB.

Multi-Party EDB: A number of secure multi-party EDB protocols [9, 33, 37, 38, 51, 62, 73, 86, 88, 90] are designed specifically for evaluating queries [1, 33, 37, 90] and carry out data analysis [9, 62, 73] based on private information that is distributed among mutually distrustful parties. Nevertheless, as noted in [10, 37, 95], due to the inherent complexity of designing multi-party protocols, operators proposed in secure multi-party EDBs are often incompatible with one another, posing significant challenges in integrating different EDB operators into a unified database framework for evaluating sophisticated SQL queries. Furthermore, distributing secrets among participating parties in MPC-based techniques generally leads to substantial requirements in communication bandwidth and interaction complexity [3, 14, 15, 37].

Outsourced EDB: Outsourced EDBs [10, 43, 55, 71, 72, 74, 77, 95] mainly focus on securely outsourcing the storage of and computation over private data to an untrusted cloud server. Most outsourced EDBs are constructed over multiple cryptographic primitives, such as searchable encryption (SE) [43, 71, 72], order-preserving encryption (OPE) [63, 74], oblivious RAM (ORAM) [22, 39, 41], partially homomorphic encryption (PHE) [43, 71, 72, 74], and fully homomorphic encryption [10, 77, 95]. However, as mentioned in [10, 95], earlier EDB constructions face challenges in terms of security (e.g., leakage-abuse attacks), efficiency (e.g., extensive communication complexity), and usability (e.g., limited query expressiveness and data analysis capability). For FHE-based outsourced EDB solutions, we provide a more detailed analysis in Section 1.2.3.

1.2.2 Encrypted Vector Searching Systems

In a vector searching system, a server holds a database consisting of high-dimensional feature vectors. To query the database, the client sends a query vector to the server to obtain the collection of feature vectors (also known as neighbors) which are "similar" to the submitted query vector. To protect data confidentiality in such vector searching systems, various EVS systems are proposed [11, 21, 32, 35, 40, 47, 48, 61, 79, 81, 87, 92, 99]. Essentially, the core of an EVS system is the privacy-preserving vector similarity search algorithm, which can be used to establish EVS query statements such as `Distance`, `Top-k`, etc. To build the vector similarity search algorithm, a variety of primitives including locality-sensitive hashing (LSH) [11, 79], OPE [87], Garbled Circuits (GC) [21], ORAM [22], and PHE [35, 40] are are employed. However, similar to outsourced EDBs, existing EVS systems also face the difficult trade-off between security and efficiency. Namely, while LSH and OPE based solutions are fast, such EVS systems do leak access patterns. In comparison, GC and ORAM guarantee provable security to the encrypted searching schemes at the cost of large communication overheads.

1.2.3 FHE-Based EDB and EVS

More recently, fast developments in FHE-based cryptographic primitives give rise to new designs of EDBs and EVSs. In Table 1, we summarize the main characteristics of existing FHE-based EDB and EVS frameworks. We first point out that most existing frameworks (such as HE^{3}DB [10], ArcEDB [95], and Cong *et al.* [32]) are either designed solely as an EDB or an EVS, but not both. Furthermore, the inherent noises residing in FHE ciphertexts pose an even bigger challenge against a usable hybrid EDB construction, as such noises can severely damage data precision during hybrid queries. As illustrated in Table 1, many existing works [10, 32, 42, 48, 55, 64, 69, 77, 93, 95, 99] suffer from inadequate data precision when performing non-linear FHE operations, such as homomorphic comparisons and ciphertext format conversions. Likewise, for queries that have deep multiplicative depth (e.g., sorting and logic aggregation functions), can only accomplish such tasks over a small number of items (i.e., bounded-size) due to the severe loss of precision after a large number of consecutive FHE multiplications.

1.2.4 Quantization for FHE

Quantization is a common technique in the field of machine learning [23, 24, 36, 54, 56, 57, 65, 98] to approximate high-precision floating-point operations using integer arithmetic. Such quantization methods are primarily used to enhance computation efficiency at the cost of reduced data precision, as low-precision integers are much easier to compute than large-precision real numbers. However, when used as an encoding step in FHE-based hybrid DBs, the loss of precision induced by data quantization can be catastrophic. Therefore, one of the main goals of this work is to devise an accuracy-aware quantization framework with rigorous error analysis, such that arbitrary-precision data can be correctly processed by expressive hybrid queries.

2 Cryptographic Preliminaries

In this section, we introduce the main concepts of the RNS-CKKS scheme and the associated FHE operators in Section 2.1 and Section 2.2, respectively.

Throughout this paper, we use bold lowercase letters (e.g., $\mathbf{a}$) for vectors, tilde lowercase letters (e.g., $\tilde{a}$) for polynomials, and bold uppercase letters (e.g., $\mathbf{A}$) for matrices. We use $\mathbf{a} \ll k$ to denote a k-step cyclic left rotation of the elements in a vector $\mathbf{a}$. We use $\circ$ to represent the Hadamard product (i.e., element-wise product) between vectors and polynomials. $\mathrm{diag}(\mathbf{A})$, $\mathrm{diag}_{+k}(\mathbf{A})$, and $\mathrm{diag}_{-k}(\mathbf{A})$ depicts the extracted vector formed by the elements on the main diagonal, the k-th super-diagonal, and the k-th sub-diagonal, respectively. We use λ to denote the security parameter, P the plaintext modulus, Q the ciphertext modulus, and N the lattice dimension. Let $\mathbb{Z}_Q$ be the set of integers modulo Q, we define R, R_Q, $R_{\mathbb{Q}}$, and $R_{\mathbb{R}}$ to denote $\mathbb{Z}(X)/(X^N+1)$, $\mathbb{Z}(X)/(X^N+1)$ mod Q, $\mathbb{Q}(X)/(X^N+1)$, and $\mathbb{R}(X)/(X^N+1)$, respectively, where $S(X)$ is the univariate polynomial in X whose coefficients are from the set S.

2.1 RNS-CKKS

Different from prior EDBs [10, 77, 95], Engorgio is built exclusively upon the RNS-CKKS [25] scheme along with the series of subsequent optimizations [12, 26, 44–46, 59].

In RNS-CKKS, the ciphertexts are constructed exclusively over the ring learning with errors (RLWE) hardness assumption [66, 76], where the encryption of a vector of plaintext

messages $\tilde{m} \in R_Q$ under a secret key $\tilde{s} \in R_Q$ is given as:

$$\mathsf{RLWE}_{\tilde{s}}^{N,Q}(\tilde{m}) = (\tilde{b}, \tilde{a}) = (-\tilde{a} \cdot \tilde{s} + \Delta\tilde{m} + \tilde{e}, \tilde{a}).$$

where $\tilde{a} \in R_Q$ is chosen uniformly at random, the noise $\tilde{e}$ is sampled from some distribution χ_{noise}, and Δ is a scaling factor to protect the least significant bits of the message from the noises. Note that, the ciphertext modulus Q can be one of a sequence of positive integers $Q_0 < Q_1 < \cdots < Q_{L-1}$, where the i-th modulus step Q_i corresponds to the i-th multiplication level. In the classic RNS-CKKS scheme, after each ciphertext multiplication, the ciphertext modulus is decremented by one level (i.e., from Q_i to Q_{i-1}), When the ciphertext modulus reaches Q_0, no more multiplications can be further performed. To increase the multiplication level back to $L-1$, the bootstrapping [7,26,59] operator is executed, where the ciphertext modulus is restored from Q_0 to Q_{L-1}.

2.2 Homomorphic Operators

2.2.1 Basic Homomorphic Operators

We primarily use the following basic homomorphic operators to evaluate linear functions (e.g., polynomials) over RLWE ciphertexts.

- $+$, $-$ and $\cdot$: Ciphertext addition, subtraction and multiplication. In this work, we consider an RLWE ciphertext to be a tuple of polynomials, where additions ($+$) and multiplications ($\cdot$) are executed in a component-wise manner between ciphertexts. For example, given two RLWE ciphertexts $\mathsf{RLWE}_0 = (\tilde{b_0}, \tilde{a_0})$ and $\mathsf{RLWE}_1 = (\tilde{b_1}, \tilde{a_1})$. The addition between the two ciphertexts is defined as $\mathsf{RLWE}(\tilde{m_0} + \tilde{m_1}) = \mathsf{RLWE}(\tilde{m_0}) + \mathsf{RLWE}(\tilde{m_1}) = (\tilde{b_0} + \tilde{b_1}, \tilde{a_0} + \tilde{a_1})$. Similarly, one can define the homomorphic subtraction between two RLWE ciphertexts. Lastly, given two RLWE ciphertexts $\mathsf{RLWE}(\tilde{m_0})$ and $\mathsf{RLWE}(\tilde{m_1})$ which encrypt the plaintext polynomials $\tilde{m_0}$ and $\tilde{m_1}$, the homomorphic multiplication $\mathsf{RLWE}(\tilde{m_0}) \cdot \mathsf{RLWE}(\tilde{m_1})$ results in $\mathsf{RLWE}(\tilde{m_0} \cdot \tilde{m_1})$. More details on ciphertext multiplication process can be found in [16].
- $\mathtt{Rotate}(\mathsf{RLWE}(\tilde{m}), \mathbf{RK}, k)$: The homomorphic rotation operator. Given a ciphertext $ct = \mathsf{RLWE}(\tilde{m})$, the rotation key $\mathbf{RK}$, and an integer $k \in \mathbb{N}$, $\mathtt{Rotate}$ outputs a ciphertext $\mathsf{RLWE}(\tilde{m} \ll k)$.
- $\mathtt{Poly}_\rho(\mathsf{RLWE}(\tilde{m}))$: Polynomial evaluation over ciphertext. For some uni-variate polynomial ρ, $\mathtt{Poly}_\rho(\mathsf{RLWE}(\tilde{m}))$ denotes the homomorphic evaluation of ρ on the input ciphertext $\mathsf{RLWE}(\tilde{m})$. In other words, after applying $\mathtt{Poly}_\rho$, the output ciphertext $\mathsf{RLWE}(\rho(\tilde{m})) = \mathtt{Poly}_\rho(\mathsf{RLWE}(\tilde{m}))$ can be decrypted to obtain $\rho(\tilde{m})$.
- $\mathtt{HomGate}(\mathsf{RLWE}(\tilde{m_0}), \mathsf{RLWE}(\tilde{m_1}), \mathtt{OP})$: Logic gate evaluation over the input ciphertexts. Given two ciphertexts $\mathsf{RLWE}(\tilde{m_0})$ and $\mathsf{RLWE}(\tilde{m_1})$ encrypting Boolean polynomials $\tilde{m_0}$ and $\tilde{m_1}$, and a two-input logic gate $\mathtt{OP}$, $\mathtt{HomGate}(\mathsf{RLWE}(\tilde{m_0}), \mathsf{RLWE}(\tilde{m_1}), \mathtt{OP})$ outputs $\mathsf{RLWE}(\tilde{m}_{\mathtt{OP}})$ where $\tilde{m}_{\mathtt{OP}}[i] = \mathtt{OP}(\tilde{m_0}[i], \tilde{m_1}[i])$ and $\tilde{m}[i]$ represents the i-th coefficient of the polynomial $\tilde{m}$. The evaluation of $\mathtt{HomGate}$ can be accomplished using the homomorphic multiplication and addition operators. Taking $\mathtt{NAND}$ as an example, we have that $\mathsf{RLWE}(\tilde{m}_{\mathtt{OP}}) = (\mathbf{0}, \mathbf{1}) - \mathsf{RLWE}(\tilde{m_0}) \cdot \mathsf{RLWE}(\tilde{m_1})$. Here, $\mathbf{0}$ and $\mathbf{1}$ depict a plaintext vector whose elements are all 0's and 1's, respectively. In this way, we can evaluate arbitrary two-input logic functions such as $\mathtt{AND}$, $\mathtt{OR}$, $\mathtt{XOR}$, $\mathtt{NOT}$, etc.

2.2.2 Approximate Homomorphic Comparison

As previously noted, the RNS-CKKS scheme supports the efficient evaluation of arithmetic operations, such as additions and multiplications, between ciphertexts. However, performing logical operations, e.g., comparisons, presents to be much more challenging. To tackle this issue, approximated homomorphic comparison operators based on polynomial approximations are proposed [27,28,49,58]. Here, we use a high-degree polynomial $\rho_{\mathtt{sgn}}$ designed to approximate the $\mathtt{sgn}$ function, specified as: $\mathtt{sgn}(x) = 1$ if $x > 0$, 0 if $x = 0$, and -1 if $x < 0$. We follow the definition in [27] and describe the precision to which the polynomial $\rho_{\mathtt{sgn}}$ approximates the sign function as (Θ, α):

$$|\rho_{\mathtt{sgn}}(x) - \mathtt{sgn}(x)| < 2^{-\alpha} \; for \; x \in [-1, -\Theta] \cup 0 \cup [\Theta, 1], \tag{1}$$

where Θ and α decide the output and input precision of the sign function, and thereby the output and input precision of the comparison operator. Therefore, we say that the following two notations are equivalent: $\rho_{\mathtt{sgn}}(x)$ and $\mathtt{sgn}_{(\Theta,\alpha)}$. Equation (1) implies that due to the discontinuous nature of $\mathtt{sgn}(x)$, $x \in [-\Theta, 0) \cup (0, \Theta]$ is not taken into account when measuring the accuracy between $\mathtt{sgn}_{(\Theta,\alpha)}$ and $\mathtt{sgn}(x)$, because the error in this interval will exceed $2^{-\alpha}$. Thus, we derive the construction of the approximate homomorphic comparison for two inputs $\mathsf{RLWE}(\tilde{x}), \mathsf{RLWE}(\tilde{y})$ with precision (Θ, α) as:

$$\begin{aligned} \mathsf{RLWE}(\tilde{r}_{\tilde{x},\tilde{y}}) &= \mathtt{ApprHomComp}_{(\Theta,\alpha)}(\mathsf{RLWE}(\tilde{x}), \mathsf{RLWE}(\tilde{y})) \\ &= \frac{\mathtt{sgn}_{(\Theta,\alpha)}(\mathsf{RLWE}(\tilde{x}) - \mathsf{RLWE}(\tilde{y})) + 1}{2}, \end{aligned} \tag{2}$$

where $\tilde{r}_{\tilde{x},\tilde{y}}$ is the comparison result:

$$\tilde{r}_{\tilde{x},\tilde{y}}[i] = \begin{cases} 1 \pm 2^{-\alpha}, & \text{if } \tilde{x}[i] > \tilde{y}[i] \\ 1/2 \pm 2^{-\alpha}, & \text{if } \tilde{x}[i] = \tilde{y}[i] \\ 0 \pm 2^{-\alpha}, & \text{if } \tilde{x}[i] < \tilde{y}[i]. \end{cases} \tag{3}$$

There are many works exploring how to optimize the degree of approximate polynomials and find better ways to composite polynomials [27, 28, 49, 58]. In this paper, since we treat $\mathtt{ApprHomComp}$ as a black box component, in order to maintain generality, we employ the Chebyshev approximation [17,20,67,78], which is one of the most basic and widely used polynomial approximation techniques, to evaluate a sign function on a ciphertext. As [70] suggests, we use polynomials of orders 59-1007 for tasks of varying precision.

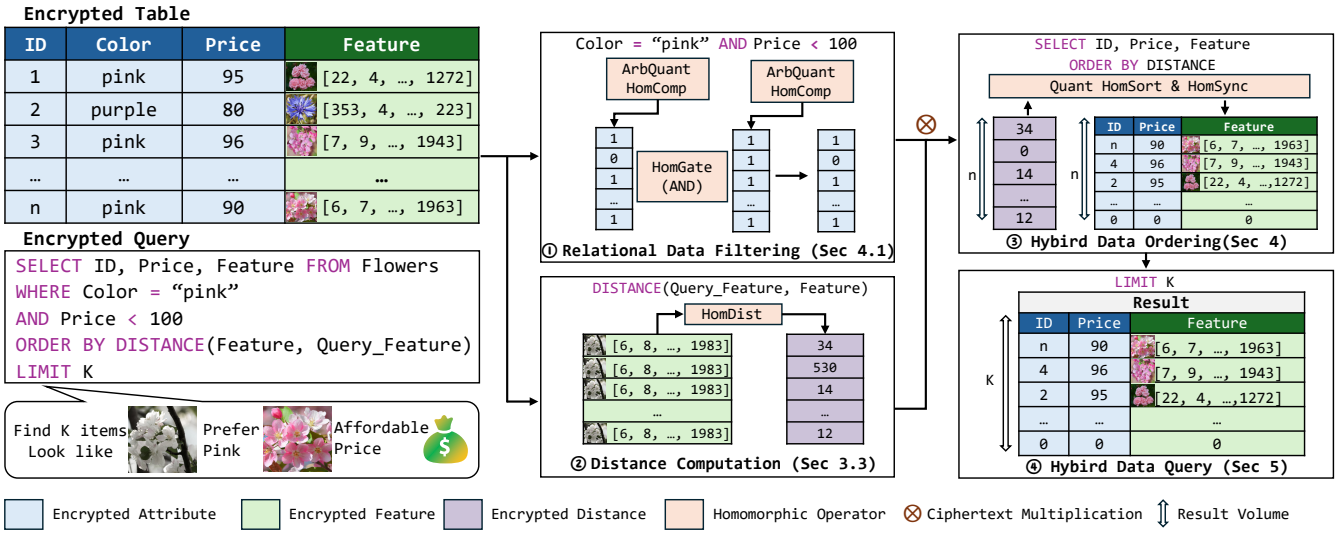


Figure 1: An example of processing a hybrid query over Engorgio. Here, we demonstrate how to run a Top-k query that combines attribute filtering with vector searching.

3 Framework Overview

In this section, we first formulate the main challenges for Engorgio in Section 3.1. Then, we provide a high-level workflow of Engorgio in Section 3.2. We then discuss the concrete constructions of the quantized FHE scheme in Section 3.3. Next, we summarize how to implement key SQL statements for the hybrid data ordering process in Section 3.4. Lastly, we present our threat model and the security analysis in Section 3.5.

3.1 Motivation and Observation

We identify two key challenges that motivate the design of Engorgio, namely, accuracy and efficiency. First, data records in hybrid EDBs can have extremely large data precision (e.g., millions of bits for image files). Thus, the underlying homomorphic comparison and sorting algorithms need to be able to handle data with arbitrarily-large precision. Second, hybrid EDBs naturally have large DB sizes. As a result, the constructed homomorphic algorithms also have to be fast, such that large-size DBs can be efficiently evaluated within a reasonable amount of time. In what follows, we explain the concrete technical challenges and observations.

Challenge and Observation 1: When comparing between ciphertexts, existing comparison methods face the fundamental trade-off between data accuracy and efficiency. For instance, the accuracy for both TFHE- and CKKS-based homomorphic comparison functions [10, 28, 58, 95] become insufficient in sorting more than 5,000 data items under a reasonably efficient parameter range. To resolve such accuracy-efficiency dilemma, we devise a new quantization framework to split large-precision data into lower-precision data segments, and execute SIMD-compatible homomorphic comparison operator over such segments to produce binary comparison outputs.

Challenge and Observation 2: Even with a sufficiently accurate comparison operator, querying a large amount of hybrid data items appears to be yet challenging due to the lack of efficient sorting algorithms. In specific, most existing sorting algorithms rely on tightly coupled comparison and permutation functions to sort data items. However, such constructions can be inefficient to sort multiple data columns. For example, in [32], applying a set of generated orders to a new data column (i.e., the order synchronization process) is as slow as re-sorting the data column. Consequently, especially when facing large hybrid DB with high cross-column synchronization needs, most existing FHE-based EDB infrastructures struggles to achieve a reasonable level of query performnace. To tackle such challenge, we propose a matrix-multiplication-based homomorphic permutation technique to accelerate the process of order synchronization, which drastically enhance the overall sorting performance over large-size hybrid EDBs.

3.2 Engorgio Workflow

Similar to prior FHE-based data management platforms [10, 77, 95], the process of running queries on Engorgio involves three main phases: client data encryption, client query encryption, and server query evaluation. Below, we provide a concise overview of the main operations involved in each phase, which are also depicted in Figure 1.

• **Client data encryption**: For every relational data table $\mathcal{T} \in \mathcal{D}$, each column t_i in $\mathcal{T}$ is divided into $J = \lceil |\mathcal{T}|_{\text{row}}/N \rceil$ pieces of size-N column chunks, and is encrypted via the `TableEncrypt` function as $[\mathcal{T}]$. Meanwhile, for each vector feature table $\mathcal{VT} \in \mathcal{D}$, each column vt_i is divided into $VJ = \lceil |\mathcal{VT}|_{\text{row}}/N \rceil$ pieces of vector column chunks, where

each vector column chunk $vc_{i,j}$ is of size N. Then, we further divide each of the vector column chunks $vc_{i,j}$ into D element blocks, assuming that the longest vector is of length D. Consequently, we have $VJ \cdot D$ pieces of size-N data blocks, which can finally be encrypted by the standard `TableEncrypt` function. The resulting encrypted vector feature table is denoted as $[\mathcal{VT}]$. Combining the relational data tables and the vector feature tables, we obtain the encrypted hybrid database $[\mathcal{D}]$ as a concatenation of vector feature table $[\mathcal{VT}]$ and relational data table $[\mathcal{T}]$. Along with the associated keys (i.e., **RK** and **BK**), $[\mathcal{D}]$ is sent to the server for cloud storage.

• **Client query encryption**: During the client query encryption phase, the client encrypts the private data in the query through the `QueryEncrypt` function. Engorgio allows users to express a hybrid query as a standard SQL format through a typical `SELECT` statement. A standard `SELECT` SQL query in Engorgio follows the form $\mathcal{Q} = (\mathcal{PA}, \mathcal{PV}, \mathsf{Agg}, \mathsf{Attr}^{\mathsf{Agg}})$, where $\mathcal{PA} = (\mathcal{P}_0, \mathcal{P}_1, ..., \mathcal{P}_{|\mathcal{PA}|-1}, \mathcal{CG}_0, \mathcal{CG}_1, ..., \mathcal{CG}_{|\mathcal{PA}|-2})$, where $\mathcal{P}_i$ represents a collection of attribute predicates over the relational DB, and $\mathcal{CG}_i$ are the logical gate functions to concatenate different predicates. In contrast to relational queries, the vector query $\mathcal{PV} = (\mathsf{Feat}, \mathcal{V}, \mathcal{OP})$ is made up of a public feature label Feat, a private feature vector $\mathcal{V}$, and a public vector operator $\mathcal{OP}$ (e.g., Top-k). Agg represents the aggregation function applied to the aggregation attribute $\mathsf{Attr}^{\mathsf{Agg}}$. To encrypt $\mathcal{Q}$, we execute `QueryEncrypt` on each of the condition values in the private predicate $\mathcal{P}_i$, as well as all elements in the private vector $\mathcal{V}$. Subsequently, we obtain the encrypted relational predicates and encrypted vector features $[\mathcal{PA}] = ([\mathcal{P}]_0, [\mathcal{P}]_1, \ldots, [\mathcal{P}]_{|\mathcal{PA}|-1}, \mathcal{CG}_0, \mathcal{CG}_1, \ldots, \mathcal{CG}_{|\mathcal{PA}|-2})$ and $[\mathcal{PV}] = (\mathsf{Feat}, [\mathcal{V}], \mathcal{OP})$, respectively. The encrypted query is then formatted as $[\mathcal{Q}] = ([\mathcal{PA}], [\mathcal{PV}], \mathsf{Agg}, \mathsf{Attr}^{\mathsf{Agg}})$.

• **Server query evaluation**: In the query evaluation phase, the server carries out the evaluation of the encrypted hybrid query $[\mathcal{Q}]$ over the encrypted hybrid database $[\mathcal{D}]$. The core steps involved in the homomorphic evaluation of the hybrid query are ① relational data filtering, ② vector distance calculation, ③ hybrid data ordering, and ④ hybrid data query. Note that, the first two steps are interchangeable depending on the exact query format. For the relational data filtering step, each encrypted attribute predicate $[\mathcal{P}]_i$ is evaluated over the corresponding encrypted attribute table $[\mathcal{T}] \in \mathcal{D}$ using the homomorphic comparison and `HomGate` functions. The filtered result is a series of ciphertexts encrypting true (a value of 1) or false (a value of 0). At the same time, `HomDist` is used to calculate the encrypted distances between the querying vector $[\mathcal{V}]$ with label Feat and the associated vector column in $[\mathcal{VT}]$ with the same attribute label. Following the relational data filtering and distance calculation, we apply a homomorphic multiplication operator to combine the filtered results with the encrypted distances, yielding a set of encrypted distances that satisfy the filtering conditions. Then the encrypted distance is ordered using `HomOrder` functions such as `HomTopK`, and the `HomSync` is employed to synchronize other columns.

3.3 Quantized FHE

In what follows, we introduce the quantized FHE scheme developed based on the RNS-CKKS scheme to effectively solve the accuracy obstacle against hybrid EDB.

3.3.1 Data Quantization and Encoding

Here, we outline the set of quantized encoding and decoding operators proposed to concretely characterize the errors homomorphically added to the plaintext messages.

• $\tilde{\mathfrak{m}}^{\zeta} \leftarrow \mathtt{QuantEncode}(\tilde{m}, \zeta, B_g)$: For a polynomial of real-valued messages $\tilde{m}(X) = \sum_{i=0}^{N-1} m_i X^i$ where $m_i \in \mathbb{R}$, given the precision parameter ζ and a quantized radix base B_g, `QuantEncode` first maps each $m_i \in \mathbb{R}$ into a rational value $m_i^{\zeta} \in \mathbb{Q}$ with precision $\zeta \in \mathbb{Z}$ as

$$m_i^{\zeta} = \mathtt{QuantEncode}(m_i)_{\mathbb{R}\to\mathbb{Q}} = \arg\min_{p_i, q_i} \left| m_i - \frac{p_i}{q_i} \right|, \quad (4)$$

where $\frac{p_i}{q_i}$ is the best rational approximation to the real number m_i up to a precision of ζ . Thus we obtain polynomial of rational-valued messages $\tilde{m}^{\zeta} = \sum_{i=0}^{N-1} m_i^{\zeta} X^i$, where $m_i^{\zeta} \in \mathbb{Q}$. After embedding m_i into the rational number m_i^{ζ} with up to ζ bits of errors, we map m_i^{ζ} into a P-bit integer $\mathfrak{m}_i^{\zeta}$ as

$$\mathfrak{m}_i^{\zeta} = \mathtt{QuantEncode}(m_i^{\zeta})_{\mathbb{Q}\to\mathbb{Z}_P} = m_i^{\zeta} 2^{\ell_{q_i}} \bmod P, \quad (5)$$

where $P = {B_g}^{\lceil \frac{\zeta}{\log B_g} \rceil}$. By conducting such mapping for each m_i^{ζ}, we say that $\tilde{\mathfrak{m}}^{\zeta}(X) = \sum_{i=0}^{N-1} \mathfrak{m}_i^{\zeta} X^i \in R_P$ is a quantized message polynomial with a precision of ζ.

• $\hat{\tilde{\mathfrak{m}}} \leftarrow \mathtt{QuantSegment}(\tilde{\mathfrak{m}}^{\zeta}, B_g)$: Note that, to encrypt data with arbitrary precision, the above approach results in an infinitely large P. Hence, for practical purposes, we apply segmentation function `QuantSegment` to $\tilde{\mathfrak{m}}^{\zeta}$ to further split the quantized message polynomial, such that each segment can fit into a fixed-precision plaintext space R_P. For some quantized message polynomial $\tilde{\mathfrak{m}}^{\zeta} = \sum_{i=0}^{N-1} \mathfrak{m}_i^{\zeta} \cdot X^i \in R_P$ with precision ζ (which can potentially extremely large such as a file), let $\mathfrak{m}_i^{\zeta} = \sum_{j=0}^{w-1} \mathfrak{m}_{\mathrm{seg}_{i,j}} {B_g}^i$, we obtain w segmented polynomials

$$\mathtt{QuantSegment}(\hat{\tilde{m}}^{\zeta}) = \hat{\tilde{\mathfrak{m}}} = \{\tilde{\mathfrak{m}}_0^{\zeta}, \tilde{\mathfrak{m}}_1^{\zeta}, \cdots, \tilde{\mathfrak{m}}_{w-1}^{\zeta}\}, \quad (6)$$

where $\tilde{\mathfrak{m}}_i^{\zeta} = \sum_{j=0}^{N-1} \mathfrak{m}_{\mathrm{seg}_{j,i}} X^j \in R_{B_g}$, and $w = \lceil \frac{\zeta}{\log B_g} \rceil$.

• $\widehat{\mathsf{QRLWE}}_{\tilde{s}}^{\zeta}(\hat{\tilde{\mathfrak{m}}}) \leftarrow \mathtt{QuantEnc}(\hat{\tilde{\mathfrak{m}}}, \tilde{s})$: Based on the quantized segmented polynomial representation, we define the quantized version an RLWE encryption as

$$\widehat{\mathsf{QRLWE}}_{\tilde{s}}^{\zeta}(\hat{\tilde{\mathfrak{m}}}) = (\mathsf{RLWE}_{\tilde{s}}(\tilde{\mathfrak{m}}_0^{\zeta}), \cdots, \mathsf{RLWE}_{\tilde{s}}(\tilde{\mathfrak{m}}_{w-1}^{\zeta})).$$

In other words, the quantized encryption of $\hat{\tilde{\mathfrak{m}}}$ under $\widehat{\mathsf{QRLWE}}$ is simply the respective encryption of $\tilde{\mathfrak{m}}_i^{\zeta}$ under the RLWE

ciphertexts. We use $\widehat{\mathsf{QRLWE}}^{\zeta}[i]$ to denote the i-th chunk $\mathsf{RLWE}_{\tilde{s}}(\tilde{\mathfrak{m}}_i^{\zeta})$ ciphertext in $\widehat{\mathsf{QRLWE}}^{\zeta}_{\tilde{s}}(\hat{\tilde{\mathfrak{m}}})$.

• $\hat{\tilde{\mathfrak{m}}} \leftarrow \texttt{QuantDec}(\widehat{\mathsf{QRLWE}}^{\zeta}_{\tilde{s}}(\hat{\tilde{\mathfrak{m}}}), \tilde{s})$: The decryption of a quantized RLWE ciphertext simply applies the CKKS decryption operator to each of the RLWE chunks $\widehat{\mathsf{QRLWE}}^{\zeta}[i]$ to get $\hat{\tilde{\mathfrak{m}}}$. Note that, applying decryption does not fully recover the original plaintext message $\tilde{\mathfrak{m}}^{\zeta}$.

• $\tilde{\mathfrak{m}}^{\zeta} \leftarrow \texttt{QuantMerge}(\hat{\tilde{\mathfrak{m}}})$: After obtaining the quantized chunks $\{\tilde{\mathfrak{m}}_i^{\zeta}\}$, we can combine the chunks using the `QuantMerge` function and get

$$\tilde{\mathfrak{m}}^{\zeta} = \texttt{QuantMerge}(\hat{\tilde{\mathfrak{m}}}) = \sum_{i=0}^{w-1} \tilde{\mathfrak{m}}_i^{\zeta} {B_g}^i \in R_P. \tag{7}$$

Essentially, `QuantMerge` is the inverse of `QuantSegment`, recombining the segmented polynomials to derive the $\tilde{\mathfrak{m}}^{\zeta}$.

• $\tilde{m}^{\zeta} \leftarrow \texttt{QuantDecode}(\tilde{\mathfrak{m}}^{\zeta})$: To further recover the original data format, we apply `QuantDecode`, the inverse function of `QuantEncode`, mapping $\tilde{\mathfrak{m}}^{\zeta}$ from R_P back to $R_{\mathbb{Q}}$:

$$\tilde{m}^{\zeta} = \texttt{QuantDecode}_{\mathbb{Z}_P \to \mathbb{Q}}(\tilde{\mathfrak{m}}^{\zeta}) = \sum_{i=0}^{N-1} \frac{\mathfrak{m}_i^{\zeta}}{2^{\ell_{q_i}}} X^i \in R_{\mathbb{Q}}. \tag{8}$$

3.4 Key Operators in Hybrid Database

In what follows, we describe the set of key homomorphic SQL operators that can be used to construct advanced SQL statements.

• $[\mathcal{T}], [\mathcal{VT}] \leftarrow \texttt{TableEncrypt}(\mathcal{T}, \mathcal{VT})$: `TableEncrypt` is constructed over the `QuantEncode`, `QuantSegment`, and `QueryEncrypt` operators to encrypt the database tables $\mathcal{T}$ and $\mathcal{VT}$ as $\widehat{\mathsf{QRLWE}}$ ciphertexts. `TableEncrypt` produces the encrypted tables $[\mathcal{T}]$ and $[\mathcal{VT}]$ corresponding to $\mathcal{T}$ and $\mathcal{VT}$, respectively.

• $[Q] \leftarrow \texttt{QueryEncrypt}(Q)$: `QueryEncrypt` encrypts a hybrid SQL query $Q = (\mathcal{PA}, \mathcal{PV}, \mathsf{Agg}, \mathsf{Attr}^{\mathsf{Agg}})$ as $\widehat{\mathsf{QRLWE}}$ ciphertexts. The result is an encrypted hybrid query $[Q] = ([\mathcal{PA}], [\mathcal{PV}], \mathsf{Agg}, \mathsf{Attr}^{\mathsf{Agg}})$.

• $[\mathcal{F}] \leftarrow \texttt{HomFilter}([\mathcal{PA}], [\mathcal{T}])$: `HomFilter` filters the encrypted table $[\mathcal{T}]$ with an encrypted attribute predicate $[\mathcal{PA}]$ and outputs a set of $\widehat{\mathsf{QRLWE}}$ ciphertexts $[\mathcal{F}]$ as results. The quantized homomorphic comparison operators `ArbQuantComp` (detailed in Section 4.1) and the `HomGate` operator mentioned in Section 2.2 are used by `HomFilter` to enhance the filtering accuracy and operator expressiveness.

• $[\mathsf{D}] \leftarrow \texttt{HomDist}([\mathcal{V}], [\mathcal{VT}])$: `HomDist` homomorphically calculates the squared Euclidean distance between the querying feature vector $[\mathcal{V}]$ and vectors stored in the encrypted vector table $[\mathcal{VT}]$ by performing element-wise homomorphic multiplication and addition in a SIMD manner. `HomDist` outputs $|\mathcal{VT}|_{\text{row}}$ encrypted distances $[\mathsf{D}]$.

• $[O] \leftarrow \texttt{HomOrder}([\mathcal{F}], [\mathsf{D}], [\mathcal{T}], [\mathcal{VT}], \mathcal{OP})$: `HomOrder` sorts and synchronizes the encrypted hybrid data in $[\mathcal{T}]$ and $[\mathcal{VT}]$ based on the encrypted distance $[\mathsf{D}]$ and attribute filtering results $[\mathcal{F}]$. The output is an encrypted ordered table $[O]$ produced according to the ordering operation specified in $\mathcal{OP}$ (e.g., `Top-k`, `Sort`, `MIN`). `HomOrder` contains two basic operators, `HomSort` and `HomSync`, which are described in Section 4.2.

• $[\mathcal{R}] \leftarrow \texttt{HomAgg}(\mathsf{Agg}, \mathsf{Attr}^{\mathsf{Agg}}, [\mathcal{T}], [O])$: `HomAgg` aggregates the column $\mathsf{Attr}^{\mathsf{Agg}}$ with the aggregation function Agg on the encrypted table $[\mathcal{T}]$ and the encrypted ordered result $[O]$. `HomAgg` outputs the encrypted aggregation result $[\mathcal{R}]$.

• $[\mathcal{R}]_i \leftarrow \texttt{HomGROUP}([Q], [\mathcal{T}], \mathsf{Attr}^{\mathsf{Group}})$: `HomGROUP` performs `GROUP BY` on the encrypted table $[\mathcal{T}]$ with the group attribute $\mathsf{Attr}^{\mathsf{Group}}$. Similar to [10, 77, 95], we issue multiple copies of $[Q]$ with equality tests (also based on `ArbQuantComp` proposed in Section 4.1) for the group attribute, and produce the grouped result $[\mathcal{R}]_i$.

3.5 Threat Model and Security

Our security goal is to protect the outsourced hybrid database $\mathcal{D}$ owned by the client $\mathcal{C}$ from the semi-honest server $\mathcal{S}$. The concrete public and private data inside the server's scope is summarized as follows.

Public Data:

• $|\mathcal{D}|$: The size of the database, i.e., the number of data (including vector) tables in $\mathcal{D}$.

• $|\mathcal{T}|_{\text{row}}$, $|\mathcal{T}|_{\text{col}}$, $|\mathcal{VT}|_{\text{row}}$, $|\mathcal{VT}|_{\text{col}}$: The number of rows as well as the number of columns in some table $\mathcal{T} \in \mathcal{D}$ (or $\mathcal{VT} \in \mathcal{D}$).

• $|Q|$: The number of filtering predicates in a SQL query.

• $\mathcal{CG}$: The logic connection (e.g., AND, OR) between the filtering predicates in a SQL query.

• $\mathsf{Attr}, |\mathsf{Attr}|, w_{Attr}$: The attribute label (e.g., gender, date), the range of the attribute (e.g., $|\mathsf{Month}| = 12$), and the number of quantized segments for this attribute.

• $\mathsf{Feat}, |\mathsf{Feat}|, w_{Feat}$: The feature label (e.g., picture), the length of vectors under the particular feature label (e.g., $|\mathsf{picture}| = 1024$ for a 32×32-pixel picture vector), and the number of quantized segments for this feature.

• $\mathcal{OP}$: The data ordering operators and the associate parameters, including `Top-k`, `Sort`, `MIN`, and `MAX`.

• The aggregation functions (e.g., `SUM`, `COUNT`) in a query.

Private Data:

• $\mathcal{T}_{i,j}$, for $i \in |\mathcal{T}|_{\text{row}}$, $j \in |\mathcal{T}|_{\text{col}}$: Exact values of the relational data items in all $\mathcal{T} \in \mathcal{D}$.

• $\mathcal{VT}_{i,j}$, for $i \in |\mathcal{VT}|_{\text{row}}$, $j \in |\mathcal{VT}|_{\text{col}}$: Exact values of the vector feature items in all $\mathcal{VT} \in \mathcal{D}$.

• $\mathcal{P}_i \in \mathcal{PA}$, $\mathcal{V}_i \in \mathcal{PV}$, for $i \in |Q|$: The attribute and vector predicate values in the SQL query.

Security of Engorgio: Similar to that in previous works [10, 31, 43, 55, 74, 77, 95], the private data owned by the client are all encrypted using FHE. Hence, the security

of Engorgio can be directly reduced to that of the underlying FHE scheme. However, one subtle difference here is that, without executing data ordering operators, data queries with vectorized predicates (e.g., $\texttt{Distance}(\mathsf{Feat}) < 50$) are susceptible to volume leakage attacks [53], as the volume of querying result is dependent on the predicate conditions. Hence, in Engorgio, we force all vectorized queries to include a data ordering operator (e.g., min, max or Top-k). Using such an approach, the amount of exposed information from a vectorized query is strictly limited to the public parameters. In other words, in our setup, existing volume leakage attacks cannot learn any additional knowledge beyond what can already be inferred from public information. We formalize the end-to-end security guarantees of Engorgio as follows:

Theorem 1. *Given some ideal functionality $\mathfrak{F}$, Engorgio securely evaluates $\mathfrak{F}$ when instantiated with a semantically-secure homomorphic encryption scheme against semi-honest adversaries with a computational security parameter of λ.*

Here, we use the ideal functionality $\mathfrak{F}$ to capture the behavior of an ideal system while satisfying the security properties mentioned above. We refer to Appendix A for a sketch of the security proof and to the full manuscript for the complete proof.

4 Quantized Homomorphic Data Ordering

In this section, we discuss details on the key cryptographic techniques to efficiently perform arbitrary-precision encrypted data ordering in Engorgio. Our approach contains two core components: the quantized homomorphic comparison algorithm `ArbQuantComp` (Section 4.1), and the quantized homomorphic sorting algorithm `HomSort` (Section 4.2).

4.1 Quantized Homomorphic Comparison

4.1.1 Binarized Homomorphic Comparison

To solve the performance-usability dilemma discussed in Section 3.1, we propose a new binarized homomorphic comparison operator `BHomComp` capable of generating Boolean logic values for a various of comparison functions, such as $>$, $<$, $==$, $\leq$, and $\geq$. Our construction is based on the approximate sign function described by Equation (3) in Section 2.2.2, where the comparison result is ternary. To binarize Equation (3), the key idea of `BHomComp` is to construct a set of linear transformations over the approximate comparison results to extract the hidden equality relation.

The detailed algorithm for `BHomComp` is provided in Algorithm 1, given two ciphertexts $ct_{\tilde{x}}$ and $ct_{\tilde{y}}$, encrypting $\tilde{x}$ and $\tilde{y}$ respectively. Let `cmp` be the comparison function, e.g., $>$, $<$, $==$, $\leq$, or $\geq$, and β be the comparison precision parameter such that $\beta < \alpha$. `BHomComp` outputs an RLWE ciphertext $ct_{\tilde{r}}$ encrypting the polynomial of comparison results $\tilde{r}$, where $\tilde{r}[i] = 1 \pm 2^{-\beta}$ if the predicate $\tilde{x}[i]$ `cmp` $\tilde{y}[i]$ is true, and $\tilde{r}[i] = 0 \pm 2^{-\beta}$ if the predicate is false. We take $>$ as an example to walk through the procedure of Algorithm 1. On Line 1–2, we compute the approximate comparison results $ct_1 = \texttt{ApprHomComp}_{(\Theta,\alpha)}(ct_{\tilde{x}}, ct_{\tilde{y}})$ and $ct_2 = \texttt{ApprHomComp}_{(\Theta,\alpha)}(ct_{\tilde{y}}, ct_{\tilde{x}}) = 1 - ct_1$. Next, on Line 3, we compute the equality result $ct_3 = 4ct_1 \cdot ct_2$. Lastly, on Line 6, we eliminate non-binary values and output the binarized comparison result as $ct_{\tilde{r}_>} = ct_1 - \frac{1}{2}ct_3$.

Algorithm 1: Binarized Homomorphic Comparison `BHomComp`

Input : Two RLWE ciphertexts $ct_{\tilde{x}}$ and $ct_{\tilde{y}}$, comparison operator `cmp`, and precision parameter (Θ, α).

Output : An RLWE ciphertext $ct_{\tilde{r}_{\texttt{cmp}}} = \mathsf{RLWE}(\tilde{r}_{\texttt{cmp}})$, where $\tilde{r}_{\texttt{cmp}}[i] = 1 \pm 2^{-\beta}$ if $\tilde{x}[i]$ `cmp` $\tilde{y}[i]$ is True, else $\tilde{r}_{\texttt{cmp}}[i] = 0 \pm 2^{-\beta}$ and $\beta < \alpha$.

1 $ct_1 \leftarrow \texttt{ApprHomComp}_{(\Theta,\alpha)}(ct_{\tilde{x}}, ct_{\tilde{y}})$
2 $ct_2 \leftarrow (\mathbf{0}, \mathbf{1}) - ct_1$
3 $ct_3 \leftarrow 4ct_1 \cdot ct_2$
4 **switch** `cmp` **do**
5 **case** $\geq$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_1 + \frac{1}{2}ct_3$;
6 **case** $>$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_1 - \frac{1}{2}ct_3$;
7 **case** $\leq$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_2 + \frac{1}{2}ct_3$;
8 **case** $<$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_2 - \frac{1}{2}ct_3$;
9 **case** $==$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_3$;
10 **case** $\neq$ **do** $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow (\mathbf{0}, \mathbf{1}) - ct_3$;

Return : $ct_{\tilde{r}_{\texttt{cmp}}}$

While `BHomComp` can carry out different kinds of comparison functions, the output ciphertexts are constrained by the limited precision parameter β, which is smaller than the input precision α. To increase β, we need to adopt significantly larger encryption parameters as well as higher degree approximation polynomials, resulting in much slower computations. To address the above issue, we design a rectification polynomial $\rho_{\text{Rectify}}(x)$ to correct the accumulated noises in and produce accurate results. Given a pre-defined precision parameter α and an interval $[1-2^{-\beta}, 1+2^{-\beta}] \cup [0-2^{-\beta}, 0+2^{-\beta}]$, we want to generate a polynomial $\rho_{\text{Rectify}}(x)$ such that

$$\forall x \in [1-2^{-\beta}, 1+2^{-\beta}] \cup [0-2^{-\beta}, 0+2^{-\beta}] \quad (9)$$
$$|\rho_{\text{Rectify}}(x) - \lceil x \rfloor| < 2^{-\alpha}.$$

Based on the equioscillation theorem [67], we are able to find the approximation polynomial of the rectify function. Through quantized encoding and the rectify function, we get the quantization layer. By inserting the above quantization layer after the `BHomComp`, the noise can be continuously controlled to be lower than $2^{-\alpha}$, which facilitates the design of the arbitrary-precision quantized homomorphic comparison.

Algorithm 2: Arbitrary-Precision Quantized Homomorphic Comparison `ArbQuantComp`

Input : Two QRLWE ciphertexts $ct_{\tilde{x}^\zeta}$, $ct_{\tilde{y}^\zeta}$.
Input : Comparison operator `cmp` and precision parameter (Θ, α).
Output : An RLWE ciphertext $ct_{\tilde{r}_{\texttt{cmp}}} = \mathsf{RLWE}(\tilde{r}_{\texttt{cmp}})$, where $\tilde{r}_{\texttt{cmp}}[i] = 1 \pm 2^{-\alpha}$ if $\tilde{x}^\zeta[i]$ `cmp` $\tilde{y}^\zeta[i]$ is True, else $\tilde{r}_{\texttt{cmp}}[i] = 0 \pm 2^{-\alpha}$.

1 **for** $i = 0$ **to** $w-1$ **do**
2 $\quad ct_{\tilde{r}_{\texttt{cmp},i}} \leftarrow \texttt{BHomComp}(ct_{\tilde{x}^\zeta}[i], ct_{\tilde{y}^\zeta}[i], \texttt{cmp}, (\Theta, \alpha))$
3 $\quad ct_{\tilde{r}_{\texttt{cmp},i}} \leftarrow \rho_{\text{Rectify}}(ct_{\texttt{cmp},i})$
4 $\quad ct_{\tilde{r}_{==,i}} \leftarrow \texttt{BHomComp}(ct_{\tilde{x}^\zeta}[i], ct_{\tilde{y}^\zeta}[i], ==, (\Theta, \alpha))$
5 $\quad ct_{\tilde{r}_{==,i}} \leftarrow \rho_{\text{Rectify}}(ct_{==,i})$
6 $ct_{\tilde{r}_0} \leftarrow ct_{\tilde{r}_{\texttt{cmp},0}}$
7 **for** $j = 1$ **to** $w-1$ **do**
8 $\quad ct_{\tilde{r}_j} \leftarrow \texttt{BatchHomMux}(ct_{\tilde{r}_{j-1}}, ct_{\tilde{r}_{\texttt{cmp},j}}, ct_{\tilde{r}_{==,j}})$
9 $ct_{\tilde{r}_{\texttt{cmp}}} \leftarrow ct_{\tilde{r}_{w-1}}$
Return : $ct_{\tilde{r}_{\texttt{cmp}}}$

4.1.2 Arbitrary-Precision Comparison

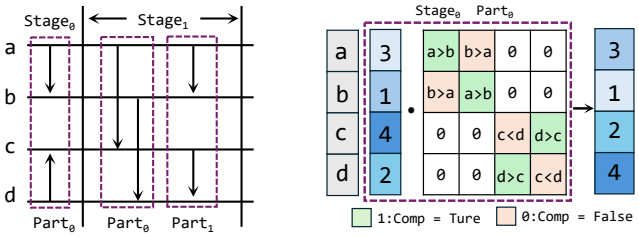
To achieve arbitrary-precision comparison without incurring heavy computational burdens, we extend the `BHomComp` operator from RLWE to $\widehat{\mathsf{QRLWE}}$. We derive a new divide-and-conquer strategy to construct the arbitrary-precision homomorphic comparison operator `ArbQuantComp` over the $\widehat{\mathsf{QRLWE}}$ ciphertexts as defined in Section 3.3.1.

In Algorithm 2, we provide the concrete algorithmic construction of `ArbQuantComp`. Given two ciphertexts $ct_{\tilde{x}^\zeta}$ and $ct_{\tilde{y}^\zeta}$ encrypting $\hat{\mathfrak{x}}$ and $\hat{\mathfrak{y}}$ as inputs, where $\hat{\mathfrak{x}}$ and $\hat{\mathfrak{y}}$ contains w quantized segmentations of $\tilde{x}^\zeta$ and $\tilde{y}^\zeta$. `cmp` is the comparison function. `ArbQuantComp` outputs an RLWE ciphertext $ct_{\tilde{r}_{\texttt{cmp}}}$ encrypting the batched comparison result polynomial $\tilde{r}$, where for each of the i-th slot in $\tilde{r}$, $\tilde{r}[i] = 1 \pm 2^{-\alpha}$ if the predicate $\tilde{x}^\zeta[i]$ `cmp` $\tilde{y}^\zeta[i]$ holds true, and $\tilde{r}[i] = 0 \pm 2^{-\alpha}$ otherwise. We use the strictly larger than function $>$ as an example to explain Algorithm 2 step-by-step. First, by definition, each of $ct_{\tilde{x}^\zeta}$ and $ct_{\tilde{y}^\zeta}$ consists w RLWE ciphertexts. Hence, to carry out the large comparison between $ct_{\tilde{x}^\zeta}$ and $ct_{\tilde{y}^\zeta}$, we first perform comparisons over each of the RLWE chunk in $ct_{\tilde{x}^\zeta}$ and $ct_{\tilde{y}^\zeta}$ via the `BHomComp` operator. To implement such comparison, on Line 1–5 in Algorithm 2, we apply the `BHomComp` to the individual RLWE ciphertexts in $ct_{\tilde{x}^\zeta}[i]$ and $ct_{\tilde{y}^\zeta}[i]$ along with the precision-rectifying polynomial ρ_{Rectify}. Next, upon obtaining the w result polynomials $\{ct_{\tilde{r}_{>,0}}, ct_{\tilde{r}_{>,1}}, \ldots, ct_{\tilde{r}_{>,w-1}}\}$, we merge the individual comparison results to produce a single RLWE ciphertext representing the final comparison outcome $ct_{\tilde{r}_>}$. Hence, on Line 6–9, we merge the comparison results $ct_{\tilde{r}_{>,i}} = \mathsf{RLWE}(\tilde{r}_{>,i})$ with the equality test results $ct_{\tilde{r}_{==,i}} = \mathsf{RLWE}(\tilde{r}_{==,i})$. In the merging process, we make use of a batched homomorphic multiplexer (MUX) operator `BatchHomMux`, which perform the following computation

$$\begin{aligned} ct_{\tilde{r}_j} &= \texttt{BatchHomMux}(ct_{\tilde{r}_{j-1}}, ct_{\tilde{r}_{>,j}}, ct_{\tilde{r}_{==,j}}) \\ &= ct_{\tilde{r}_{>,j}} + ct_{\tilde{r}_{==,j}} \cdot (ct_{\tilde{r}_{j-1}} - ct_{\tilde{r}_{>,j}}). \end{aligned} \tag{10}$$

Here, $ct_{\tilde{r}_j}$ encrypts a polynomial $\tilde{r}_j$ where $\tilde{r}_j[i]$ equals $\tilde{r}_{j-1}[i]$ when $\tilde{r}_{==,j}[i] == 1$ and $\tilde{r}_{>,j}[i]$ otherwise, i.e., $\tilde{r}_j[i] = (\tilde{r}_{==,j}[i] == 1)\ ?\ \tilde{r}_{j-1}[i] : \tilde{r}_{>,j}[i]$. Essentially, `BatchHomMux` homomorphically selects between $ct_{\tilde{r}_{j-1}}$ and $ct_{\tilde{r}_{>,j}}$ based on the value of $ct_{\tilde{r}_{==,j}}$ in a SIMD manner. Line 6–9 in Algorithm 2 is simply iterating over each of the segmented comparison result $ct_{\tilde{r}_{>,j}}$ from small to large, and check if the comparison between the j-th data chunks determines the final comparison result. In other words, considering the case $w = 2$, the comparison $\tilde{x}^\zeta > \tilde{y}^\zeta$ can be computed as $(\hat{\mathfrak{x}}[1] == \hat{\mathfrak{y}}[1])\ ?\ (\hat{\mathfrak{x}}[0] > \hat{\mathfrak{y}}[0]) : (\hat{\mathfrak{x}}[1] > \hat{\mathfrak{y}}[1])$, where $\hat{\mathfrak{x}}[1]$ and $\hat{\mathfrak{y}}[1]$ encrypt the most significant bits of $\tilde{x}$ and $\tilde{y}$, respectively.

Note that, as specified in Equation (6), the data precision for each of the segmented plaintext chunk in the input QRLWE ciphertexts is B_g. In contrast, the precision of the resulting RLWE ciphertext from `ArbQuantComp` is α. Hence, we set $\alpha > 2B_g$ to ensure that the `ArbQuantComp` operator generates outputs with the same precision level as the inputs.



(a) Bitonic sorting network (b) Matrix-based permutation

Figure 2: Examples of (a) a 4-element bitonic sorting network and (b) the associated matrix-based permutation process.

4.2 Quantized Homomorphic Sorting

4.2.1 The Basic Sorting Scheme

Based on the quantized comparison operators, we propose a new two-stage sorting framework to efficiently execute the bitonic sorting algorithm [8] over FHE ciphertexts. Our `HomSort` operator construction follows the bitonic algorithm [8], where the sorting process proceeds in terms of parts and stages. As shown in Figure 2a, each *stage* in the bitonic sorting network contains multiple parts. For a table with a row size of $|\mathcal{T}|_{\text{row}}$, each part performs $|\mathcal{T}|_{\text{row}}/2$ sets of comparisons and data swaps. Completing all the parts in the i-th stage in the sorting network produces $|\mathcal{T}|_{\text{row}}/2^{i+1}$ partially sorted subsequences, each of length 2^{i+1}. Based on the architecture of the bitonic network, we divide our sorting procedure into two steps: the order generation (i.e., the comparison) step and the order application (i.e., the permutation)

Algorithm 3: Order Generation `HomOrdGen`

Input : A QRLWE ciphertext $ct_{\tilde{x}}$, rotate key **RK**, precision parameter (Θ, α), table size $|\mathcal{T}|_{\text{row}}$, stage s and part p.

Output : An RLWE ciphertext $ct_{\texttt{cmp}_{s,p}}$, which encrypts the comparison result of stage s, part p.

1 **for** $i = 0$ **to** $|\mathcal{T}|_{\text{row}}$ **step** 2^{s+2} **do**
2 $\quad$ **Mask**$[i : i+2^{s+1}] \leftarrow 1$
3 $\quad$ **Mask**$[i+2^{s+1} : i+2^{s+2}-1] \leftarrow -1$
4 **for** $i = 0$ **to** $|\mathcal{T}|_{\text{row}}$ **step** 2^{s-p+1} **do**
5 $\quad$ **Eli**$[i : i+2^{s-p}] \leftarrow 1$
6 $\quad$ **Eli**$[i+2^{s-p} : i+2^{s-p+1}-1] \leftarrow 0$
7 **for** $i = 0$ **to** $w-1$ **do**
8 $\quad$ $ct_{\text{mask}}[i] \leftarrow ct_{\tilde{x}}[i] \cdot$ **Mask**
9 $\quad$ $ct_{\text{rot}}[i] \leftarrow \texttt{Rotate}(ct_{\text{mask}}[i], \mathbf{RK}, 2^{s-p})$
10 $ct_{\texttt{cmp}_{s,p}} \leftarrow \texttt{ArbQuantComp}(ct_{\text{mask}}, ct_{\text{rot}}, >, (\Theta, \alpha)) \cdot$ **Eli**

Return : $ct_{\texttt{cmp}_{s,p}}$

Algorithm 4: Order Application `HomOrdApp`

Input : A QRLWE ciphertext $ct_{\tilde{x}}$, an encoded permutation matrix $\mathbf{M}_{s,p}$, table size $|\mathcal{T}|_{\text{row}}$, stage s, part p, and rotate key **RK**.

Output : A QRLWE ciphertext ct_{res}, which encrypts the sort result of stage s, part p.

1 **for** $i = 0$ **to** $w-1$ **do**
2 $\quad$ $ct_{\text{sup}}[i] \leftarrow \texttt{Rotate}(ct_{\tilde{x}}[i], \mathbf{RK}, |\mathcal{T}|_{\text{row}} - 2^{s-p})$
3 $\quad$ $ct_{\text{sub}}[i] \leftarrow \texttt{Rotate}(ct_{\tilde{x}}[i], \mathbf{RK}, 2^{s-p})$
4 $\quad$ $ct_{\text{res}}[i] \leftarrow$
$\quad\quad ct_{\text{sup}}[i] \cdot \mathbf{M}_{s,p}[0] + ct_{\tilde{x}}[i] \cdot \mathbf{M}_{s,p}[1] + ct_{\text{sub}}[i] \cdot \mathbf{M}_{s,p}[2]$

Return : ct_{res}

Algorithm 5: Homomorphic Sort `HomSort`

Input : An unsorted QRLWE ciphertext $ct_{\tilde{x}}$, rotate key **RK**, stage s, part p, table size $|\mathcal{T}|_{\text{row}}$, and precision parameter (Θ, α).

Output : A QRLWE ciphertext ct_{res}, which encrypts a sorted column.

1 $ct_{\text{res}} \leftarrow ct_{\tilde{x}}$
2 **for** $s = 0$ **to** $\log_2(|\mathcal{T}|_{\text{row}})$ **do**
3 $\quad$ **for** $p = 0$ **to** s **do**
4 $\quad\quad$ $ct_{\texttt{cmp}_{s,p}} \leftarrow$
$\quad\quad\quad \texttt{HomOrdGen}(ct_{\text{res}}, \mathbf{RK}, (\Theta, \alpha), |\mathcal{T}|_{\text{row}}, s, p)$
5 $\quad\quad$ $\mathbf{M}_{s,p} \leftarrow \texttt{MEncode}(ct_{\texttt{cmp}_{s,p}}, \mathbf{RK}, |\mathcal{T}|_{\text{row}}, s, p)$
6 $\quad\quad$ $ct_{\text{res}} \leftarrow$
$\quad\quad\quad \texttt{HomOrdApp}(ct_{\text{res}}, \mathbf{M}_{s,p}, |\mathcal{T}|_{\text{row}}, s, p, \mathbf{RK})$

Return : ct_{res}

step. Here, we consider the case where $|\mathcal{T}|_{\text{row}} \leq N$, while the extension for $|\mathcal{T}|_{\text{row}} > N$ will be discussed in Section 5.1. In what follows, we take Figure 2 as an example to explain the overall sorting procedure.

• **Order Generation Step**: Taking an unsorted array $\mathbf{x} = [a, b, c, d]$ as example, according to the bitonic sorting network illustrated in Figure 2a, we need to compute $a > b$ and $d > c$ in the zeroth part of the zeroth stage ($part_0$, $stage_0$). Hence, suppose that $\mathbf{x}$ is encrypted as $ct_{\tilde{x}}$, we derive the `HomOrdGen` operator depicted in Algorithm 3. As specified on Line 1–6 of Algorithm 3, we first initialize the mask and elimination vectors **Mask** and **Eli**. Then, on Line 8, we multiply $ct_{\tilde{x}}$ by the mask vector **Mask** $= [1, 1, -1, -1]$ and get $ct_{\tilde{x}_{\text{Mask}}}$, which encrypts $[a, b, -c, -d]$. Next, we compute $\texttt{Rotate}(ct_{\tilde{x}}[i] \cdot \mathbf{Mask}, \mathbf{RK}, 1)$ to obtain $ct_{\tilde{x}_{\text{rot}}}$,which encrypts $[b, -c, -d, a]$ (Line 9). Lastly on Line 10, we conduct the $>$ comparison between $ct_{\tilde{x}_{\text{mask}}}$ and $ct_{\tilde{x}_{\text{rot}}}$, followed by a coefficient-wise multiplication by an elimination vector **Eli** $= [1, 0, 1, 0]$. After the multiplication, we obtain the comparison result $ct_{\texttt{cmp}_{s,p}}$ that encrypts $[a > b, 0, d > c, 0]$. Note that the comparison result $\texttt{cmp}_{s,p}$ is essentially a sequence of relative orders between a, b, c and d. Hence, we refer to this step as the order generation step.

• **Order Application Step**: After order generation, we need to apply the orders over the input vector via element-wise permutation. Unlike prior works [10, 32, 64, 95], we use homomorphic vector-matrix multiplication to homomorphically permute the plaintext elements in the ciphertexts. In Figure 2b, we show an example of our matrix-based permutation technique. To begin with, we encode the comparison results into the corresponding positions of the permutation matrix using the `MEncode` operator (the concrete encoding procedures are detailed in Appendix B.1). Then, we only need to perform a simple inner product between an unsorted input vector and the permutation matrix, i.e., the `HomOrdApp` operator, to swap the elements in the input vector and produce the sorted output. The detailed algorithm for `HomSort` is provided in Algorithm 5. Given an unsorted QRLWE ciphertext $ct_{\tilde{x}}$, `HomSort` iterate through the each of the $\log_2(|\mathcal{T}|_{\text{row}})(\log_2(|\mathcal{T}|_{\text{row}}) + 1)/2$ parts of the bitonic algorithm (Line 2–3). The main loop body calls `HomOrdGen`, `MEncode`, and `HomOrdApp` on Line 4–6 repetitively, resulting in the sorted QRLWE ciphertext ct_{res}. After sorting one specific column, we can also synchronize the orders (or intermediate orders) of the sorted column to other data columns. In particular, we can extract all the permutation matrices $[\mathbf{M}_{0,0}, \mathbf{M}_{1,0}, \cdots, \mathbf{M}_{\log_2(|\mathcal{T}|_{\text{row}})-1, \log_2(|\mathcal{T}|_{\text{row}})-1}]$ during the sorting process, and execute `HomOrdApp` on the target columns to synchronize the data orders. The exact synchronization algorithm is detailed in Appendix B.2.

Remark: The advantage of using vector-matrix multiplication for order application is that once the permutation matrix is obtained, we can directly use the permutation matrix to synchronize the order to other data columns through `HomSync`. However, the homomorphic vector-matrix multiplication has complexity $O(|\mathcal{T}|^2_{\text{row}})$ for permuting $|\mathcal{T}|_{\text{row}}$ items, which can be slow. Therefore, in the following section, we introduce

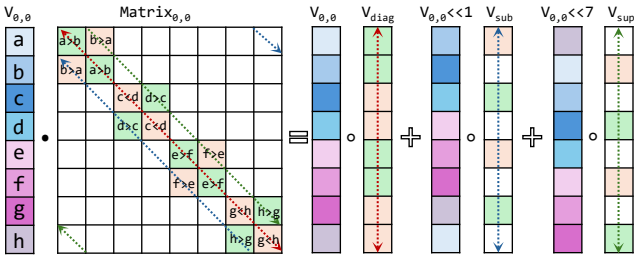


Figure 3: Demonstrations of (a) the symmetric sparsity matrix encoding and (b) the matrix-based permutation process.

a new matrix encoding method to reduce the permutation complexity from $O(|\mathcal{T}|^2_{\text{row}})$ to $O(|\mathcal{T}|_{\text{row}})$.

4.2.2 Matrix-Based Fast Permutation

We note that, when permuting the to-be-sorted vectors, the permutation matrix has the following two properties as shown in Figure 3: i) symmetric along the diagonal, and ii) only the main diagonal, one sub-diagonal, and one super-diagonal contain valid order information.

Based on the above observations, we propose a fast permutation algorithm `HomOrdApp` to homomorphically permute a vector based on the symmetric sparsity matrix encoding. Figure 3 illustrates the permutation matrix $\mathbf{M}_{0,0}$ of $(stage_0, part_0)$ where the input vector $\mathbf{v}_{0,0}$ has a length of eight. We can see that, only the elements on the main diagonal and the dotted lines parallel to the diagonal contain the actual comparison results, while the rest of the positions are all zeroes. Subsequently, we encode the elements on these three lines as $\mathbf{v}_{\text{diag}} = \text{diag}(\mathbf{M}_{0,0})$, $\mathbf{v}_{\text{sub}} = [\text{diag}_{-1}(\mathbf{M}_{0,0}), 0]$, and $\mathbf{v}_{\text{sup}} = [0, \text{diag}_{+1}(\mathbf{M}_{0,0})]$. Then, the permutation of $\mathbf{v}_{0,0}$ based on $\mathbf{M}_{0,0}$ is given by

$$\mathbf{v}_{0,0} \cdot \mathbf{M}_{0,0} = \mathbf{v}_{0,0} \circ \mathbf{v}_{\text{diag}} + (\mathbf{v}_{0,0} \ll 1) \circ \mathbf{v}_{\text{sub}} + (\mathbf{v}_{0,0} \ll 7) \circ \mathbf{v}_{\text{sup}}.$$

More concretely, as shown in Algorithm 4, to permute a QRLWE ciphertext $ct_{s,p}$ containing w RLWE ciphertexts of length $|\mathcal{T}|_{\text{row}}$ based on an encoded permutation matrix $\mathbf{M}_{s,p}$, we invoke $2w$ homomorphic rotations on Line 1-3 and $3w$ homomorphic multiplications on Line 4 to carry out the homomorphic permutation.

4.2.3 Complexity Analysis

Here, we compare the concrete SQL operation costs of Engorgio against HE^{3}DB [10] and ArcEDB [95]. As shown in Table 2, for most SQL statements, Engorgio can reduce the computation costs by a factor of N due to the use of SIMD packing. As a result of both SIMD packing and sparse matrix encoding, when all data items can be packed into a single ciphertext, the number of ciphertext multiplications remains constant (i.e., $O(1)$) with respect to the number of data items in Engorgio. Furthermore, by solely relying on the RNS-CKKS scheme, Engorgio does not need to apply the costly ciphertext conversion operations as in [10, 95], significantly improving the performance of most filter-aggregation queries. In Section 6.2, we quantitatively assess the performance gain of Engorgio against [10, 95].

5 Hybrid Data Querying Optimization

In this section, we propose several optimization techniques when applying the sorting operator presented in Section 4 in hybrid data querying.

5.1 Full Table Scan Query

When executing hybrid queries over data tables $\mathcal{T}$ whose row size exceed the number of SIMD slots in the FHE encryption scheme (i.e., $|\mathcal{T}|_{\text{row}} > N$), existing works face challenges either in terms of limited precision [10, 32] or incompatible operator design [48, 95]. To address both issues, we devise an extension to our matrix-based homomorphic sorting scheme proposed in Section 4.2 to act over multiple ciphertexts. Given a set of QRLWE ciphertexts $\{ct_0, ct_1, \cdots, ct_l\}$ that encrypt one long unsorted array of length (padded to) $N \cdot l$, the idea is to break the size-$N \cdot l$ permutation into smaller N-size permutations through matrix decomposition. Since the size-$(N \cdot l)^2$ permutation matrix $\mathbf{M}_{s,p}$ can be easily decomposed into l size-N permutation matrices $\{\mathbf{M}_0, \mathbf{M}_1, \cdots, \mathbf{M}_l\}$ along its main diagonal. For example, if $l = 2$, we have

$$\begin{aligned}
&[ct_0, ct_1] \cdot \mathbf{M}_{s,p} = [ct_0, ct_1] \cdot [\text{diag}(\mathbf{M}_0), \text{diag}(\mathbf{M}_1)] + \\
&[ct_0, ct_1] \ll (2^{s-p}) \cdot [\mathbf{0}, \text{diag}_{-2^{s-p}}(\mathbf{M}_0), \text{diag}_{-2^{s-p}}(\mathbf{M}_1)] + \\
&[ct_0, ct_1] \ll (N - 2^{s-p}) \cdot [\text{diag}_{2^{s-p}}(\mathbf{M}_0), \text{diag}_{2^{s-p}}(\mathbf{M}_1), \mathbf{0}].
\end{aligned}$$

Due to space limitations, more details on the unbounded-size sorting algorithms are outlined in Appendix B.3.

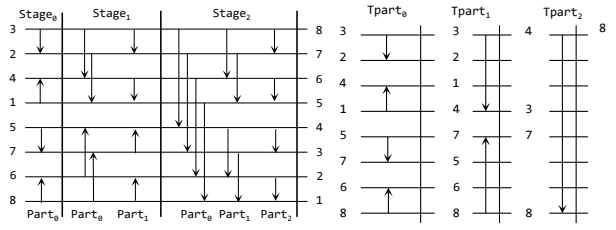
5.2 Batched Query

In contrast to the case where $|\mathcal{T}|_{\text{row}} > N$ as discussed above, we also see real-world databases with table sizes much less than the number of SIMD slots, i.e., $|\mathcal{T}|_{\text{row}} \ll N$. In such case we can batch multiple queries into a single ciphertext to enhance the overall efficiency. For example, suppose that we wish to execute t queries on a table of d rows where $d \ll N$. We can batch the t queries (either relational or vectorized) into a single query as long as $d \cdot t < N$. Although batched filtering and distance computations are relatively easy to achieve, batched sorting can be more challenging due to the need of early termination in the bitonic sorting network. Specifically, to batch sorting d sets of filtered results, we need to first encode the i-th hybrid filter result into the $i \times d$-th position of the batched input vector. Then, we perform the

Table 2: Operation costs comparison for each homomorphic SQL statements.

SQL$^+$	Method	# Comparison	# Gate	# Conversion	# Multiplication	# Rotate
SELECT	ArcEDB [95]	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert Q\rvert$	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot (\lvert Q\rvert - 1)$	$\log(P)\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert Q\rvert^*$	None	None
	HE^{3}DB [10]	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert Q\rvert$	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot (\lvert Q\rvert - 1)$	None	None	None
	This Work	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert Q\rvert / N$	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot (\lvert Q\rvert - 1)/N$	None	None	None
MIN/MAX	ArcEDB [95]	$\log(P)(\lvert\mathcal{T}\rvert_{\text{row}} - 1)$	None	$\log(P)(2\lvert\mathcal{T}\rvert_{\text{row}} - 1)$	$\log(P)(2\lvert\mathcal{T}\rvert_{\text{row}} - 1)$	None
	HE^{3}DB [10]	$\lvert\mathcal{T}\rvert_{\text{row}} - 1$	None	$2\lvert\mathcal{T}\rvert_{\text{row}} - 1$	$2\lvert\mathcal{T}\rvert_{\text{row}} - 1$	None
	This Work	$(\lvert\mathcal{T}\rvert_{\text{row}} - 1)/N$	$(\lvert\mathcal{T}\rvert_{\text{row}} - 1)/N$	None	$4(\lvert\mathcal{T}\rvert_{\text{row}} - 1)/N$	$3(\lvert\mathcal{T}\rvert_{\text{row}} - 1)/N$
GROUP BY	ArcEDB [95]	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert\mathsf{Attr}^{\mathsf{Group}}\rvert$	None	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert\mathsf{Attr}^{\mathsf{Group}}\rvert$	None	None
	HE^{3}DB [10]	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert\mathsf{Attr}^{\mathsf{Group}}\rvert$	None	None	None	None
	This Work	$\lvert\mathcal{T}\rvert_{\text{row}} \cdot \lvert\mathsf{Attr}^{\mathsf{Group}}\rvert / N$	None	None	None	None
ORDER BY	ArcEDB [95]	$\log(P)(\lvert\mathcal{T}\rvert_{\text{row}}^2 - \lvert\mathcal{T}\rvert_{\text{row}})/2$	$2\log(P)\lvert\mathcal{T}\rvert_{\text{row}}$	$\log(P)\lvert\mathcal{T}\rvert_{\text{row}}$	$\lvert\mathcal{T}\rvert_{\text{row}}^2 - \lvert\mathcal{T}\rvert_{\text{row}}$	$\lvert\mathcal{T}\rvert_{\text{row}} - 1$
	HE^{3}DB [10]	$\lvert\mathcal{T}\rvert_{\text{row}} \log^2 \lvert\mathcal{T}\rvert_{\text{row}}$	None	$\lvert\mathcal{T}\rvert_{\text{row}} (\log^2 \lvert\mathcal{T}\rvert_{\text{row}} + 1)$	$\lvert\mathcal{T}\rvert_{\text{row}} (\log^2 \lvert\mathcal{T}\rvert_{\text{row}} + 1)$	None
	This Work	$\lvert\mathcal{T}\rvert_{\text{row}} \log^2(\lvert\mathcal{T}\rvert_{\text{row}})/N$	$\lvert\mathcal{T}\rvert_{\text{row}} \log^2(\lvert\mathcal{T}\rvert_{\text{row}})/N$	None	$4\lvert\mathcal{T}\rvert_{\text{row}} \log^2(\lvert\mathcal{T}\rvert_{\text{row}})/N$	$3\lvert\mathcal{T}\rvert_{\text{row}} \log^2(\lvert\mathcal{T}\rvert_{\text{row}})/N$

$\lvert Q\rvert$, $\lvert\mathcal{T}\rvert_{\text{row}}$, $\lvert\mathsf{Attr}^{\mathsf{Group}}\rvert$, $\log(P)$ are publicly known to the server. * Required when the SELECT condition involves inner table data or aggregated results.
$^+$ The complexities of SUM, COUNT and AVG operators are not listed because they are consistent in HE^{3}DB, ArcEDB and this work.



(a) 8-input sorting network (b) Top-1 of 8

Figure 4: The conceptual illustrations of (a) a full 8-element sorting network and (b) the pruned Top-1 network.

aforementioned HomSort algorithm on the batched input vector but early terminate the sorting process after completing the $(\log_2(d) - 1)$-th stage. Due to the inherent property of the bitonic algorithm, after $\log_2(d) - 1$ stages, the batched input vector is partially sorted at every d intervals, which is precisely the desired sorted result. We provide a detailed BatchedSort algorithm in Appendix B.4.

5.3 Top-k Query

As mentioned in [50], it is generally the case that database clients only care about the most relevant k results (i.e., Top-k) to a particular query. Thus, we propose an optimization for implementing the Top-k query statement over the HomSort operator inspired by [80].

As mentioned in Section 5.2, due to the inherent property of the bitonic algorithm, the intermediate ciphertexts at the $\log_2(d)$-th stage contain $\lvert\mathcal{T}\rvert_{\text{row}}/2d$ partially ordered sequences of length d. Therefore, we can cut off certain parts in the sorting network to improve the performance of the Top-k statement. In Figure 4b, we show an example of selecting the Top-1 out of 8 input elements. The observation here is that, we can prune the 8-input sorting network in Figure 4a to only sort the elements that have the potential to become the Top-1 element. Roughly speaking, to select the Top-k from $\lvert\mathcal{T}\rvert_{\text{row}}$ elements, we first perform the $\log_2(k)$ complete storing stages over all the $\lvert\mathcal{T}\rvert_{\text{row}}$ elements. Then, we get $\lvert\mathcal{T}\rvert_{\text{row}}/2k$ sorted blocks, each of length $2k$. Next, we execute the $\log_2(k)$-th stage for $\log_2(\lvert\mathcal{T}\rvert_{\text{row}}) - \log_2(k) - 1$ times. During t-th execution of the $\log_2(k)$-th stage, we adjust the rotation step of $part_0$ to $2^{\log_2(k)} + (2^{t+1} - 1) \cdot 2k$. Subsequently, we get the top k out of the $\lvert\mathcal{T}\rvert_{\text{row}}$ elements. We observe that the complexity of the above homomorphic Top-k algorithm is $O(\lvert\mathcal{T}\rvert_{\text{row}} \log^2 k)$ for an input sequence of $\lvert\mathcal{T}\rvert_{\text{row}}$ elements. Note that, since it is generally the case that $k \ll \lvert\mathcal{T}\rvert_{\text{row}}$, we obtain a complexity reduction of nearly $\log^2 \lvert\mathcal{T}\rvert_{\text{row}}$ when compared to the unmodified bitonic algorithm (which has a complexity of $O(\lvert\mathcal{T}\rvert_{\text{row}} \log^2 \lvert\mathcal{T}\rvert_{\text{row}})$). The concrete homomorphic Top-k algorithm is explained in Appendix B.5.

6 Evaluation

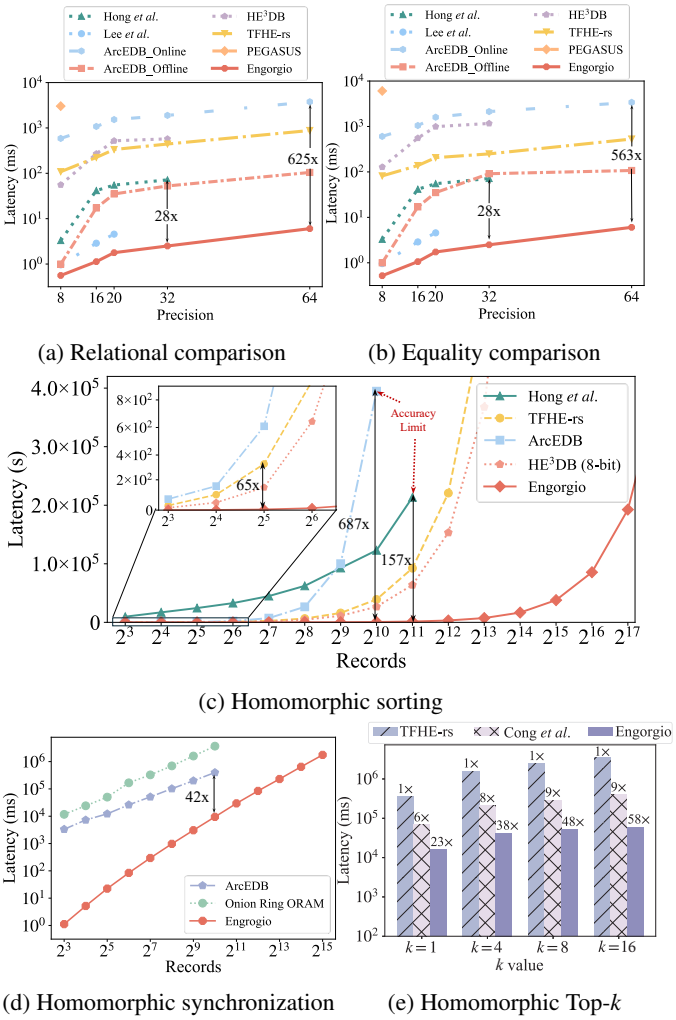
In evaluating Engorgio, we wish to answer the following two main research questions (RQs).

•**RQ1**: How efficient are the individual components of Engorgio compared to state-of-the-art (SOTA) methods?

•**RQ2**: How does Engorgio perform in relational, vector, and hybrid SQL queries?

6.1 Experiment Setup

We implement Engorgio based on OpenFHE [6] using C++17 and compiled with GCC 10.2.0. We set the lattice dimension $D = 2^{17}$, the ciphertext slot size $N = D/2 = 2^{16}$, and the largest ciphertext modulus Q_L to be a 3,516-bit integer. The secret keys are drawn from a ternary distribution as specified in [29]. The above parameters provide 128-bit of security level according to [2, 13]. The experiments are carried out on an Intel Xeon Gold 6226R processor with 512 GB of RAM in



(a) Relational comparison (b) Equality comparison

(c) Homomorphic sorting

(d) Homomorphic synchronization (e) Homomorphic Top-k

Figure 5: Microbenchmark results for homomorphic comparison, sorting, synchronization, and Top-k.

Table 3: Benchmark results for SQL statements over 1K rows.

SQL	HE^{3}DB [10] (s)		ArcEDB [95] (s)		**Engorgio (s)**	
`SELECT`	276.2	1×	9.9	28×	**1.2**	**230**×
`MIN/MAX`	1,219.4	10×	12,501.5	1×	**79.9**	**156**×
`GROUP BY`	377.9	2.1×	807.9	1×	**2.3**	**349**×
`ORDER BY`	26,613.1	15×	394,724.9	1×	**574.3**	**687**×

a single-thread environment. The relational part of our hybrid database is made up of the TPC-H benchmarks [34] and the vector part is made up of the data in SIFT1M dataset [52].

6.2 Microbenchmarks

To answer **RQ1**, we perform a comprehensive set of microbenchmarks to test the efficiency for each of the basic homomorphic operators in Engorgio, including data filtering, data ordering, order synchronization, and Top-k selection.

Table 4: Ciphertext size analysis of 1M rows records with different precision.

Precision	Methods	Ciphertext Size	
16-bit	ArcEDB [95]	4GB	512×
	HE^{3}DB [10]	2GB	256×
	This Work	8MB	1×
32-bit	ArcEDB [95]	8GB	512×
	HE^{3}DB [10]	2GB	128×
	This Work	16MB	1×
64-bit	ArcEDB [95]	16GB	512×
	HE^{3}DB [10]	—	—
	This Work	32MB	1×

– means the method does not support such level of precision.

• **Filtering**: Here, we compare the proposed `ArbQuantComp` operator against the best-performing existing solutions [10, 48, 58, 64, 93, 95]. As shown in Figure 5a and Figure 5b, many existing works cannot perform homomorphic comparisons with 64-bit precision. Note that, ArcEDB [95] is categorized into online and offline phase, with details provided in Appendix C.1. Notably, Engorgio is 28×–854× faster than the SOTA homomorphic comparison methods for a precision level of 32-bit or above.

• **Sorting**: We compare `HomSort` to existing solutions that support homomorphic sort [10, 48, 64, 93, 95]. We vary the unsorted vector length from 2^3 to 2^{17} with 16-bit precision. As shown in Figure 5c, due to the large noise growth, most existing methods [48, 95] are limited by the number of elements that can be sorted (i.e., the length of the unsorted vector). Meanwhile, HE^{3}DB [10] can only sort vectors with 8-bit precision elements due to the noise growth from ciphertext conversion. In contrast, Engorgio can sort vectors with length beyond 2^{16} while demonstrating 65×–687× speedup against the SOTA sorting algorithms.

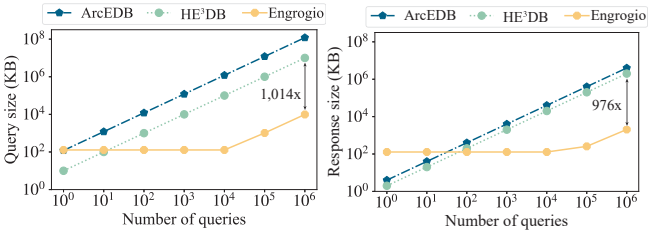
• **Synchronization and Top-k**: We test the performance of order synchronization and Top-k query. As shown in Figure 5d, Engorgio is 42×–4,595× faster than leading methods [22, 95] when the synchronizing sequence length ranges from 2^3 to 2^{15}. Similarly, for Top-k, as depicted in Figure 5e, Engorgio is 4×–58× faster than the SOTA solutions [32, 93].

• **SQL Operation**: Through Table 3, we illustrate that, to echo with the theoretical complexity analysis in Section 4.2.3, Engorgio can indeed accelerate the evaluation speeds of various homomorphic SQL statements by 8×–687×, when compared to those in [10] and [95].

6.3 SQL Benchmarks

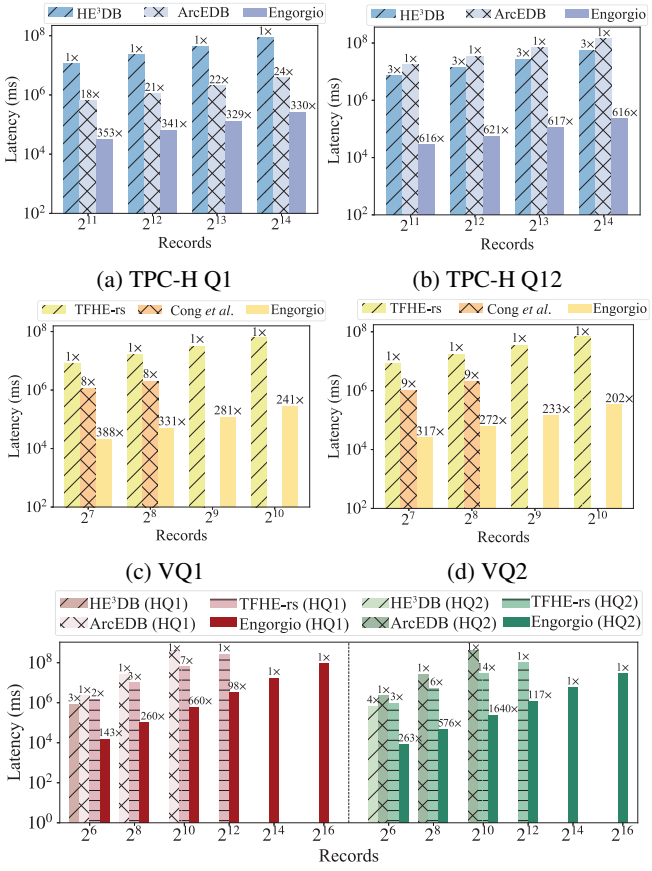
To answer the **RQ2**, we evaluated the performance of Engorgio on various end-to-end SQL queries, including relational queries, vectorized queries, and hybrid queries.

• **Relational Query**: For the relational query, we compare our results with the most recent FHE-based frameworks [10, 95]. We use queries in TPC-H benchmarks [34] to



(a) Query size (b) Response size

Figure 6: Comparisons of query and response sizes with varying numbers of concurrent TPC-H Q6 queries.



(a) TPC-H Q1 (b) TPC-H Q12

(c) VQ1 (d) VQ2

(e) Hybrid query performance comparisons

Figure 7: Relational, vectorized, and hybrid end-to-end SQL query performance comparisons with varying sizes of $|\mathcal{T}|_{\text{row}}$.

test the performance of Engorgio under varying row sizes. As shown in Figure 7a, Engorgio is 15×–20× faster than [95] and 329×–353× faster than [10] over 16-bit TPC-H Q1. As illustrated in Figure 7b, for queries like TPC-H Q12 that involve inner-table comparisons, Engorgio can be 242×–251× faster than [10] and 616×–621× faster than [95]. In terms of memory performance, as illustrated in Table 4, we can reduce the peak memory consumption by up to 512×.

• **Vectorized Query**: We construct two vectorized queries VQ1 and VQ2, which consist of vector searching, Top-k, and synchronization statements (detailed in Appendix C.2). We compare our results with the most relevant frameworks [32, 93]. As shown in Figure 7c, Engorgio is 43×–55× (resp. 30×–38×) faster than [32] and 241×–388× (resp. 202×–317×) faster than [93] over VQ1 (resp. VQ2).

• **Hybrid Query**: We construct two hybrid queries HQ1 and HQ2 (detailed in Appendix C.2). Since there is no publicly available work that support hybrid queries over EDBs, we estimate the performance of existing works [10, 93, 95] based on the supported operators. As shown in Figure 7e, Engorgio runs 68×–80× faster than [10], 80×–120× faster than TFHE-rs [93], and 142×–1,640× faster than [95] under various row sizes for HQ1 and HQ2.

• **Communication Costs**: We compare the query and response sizes with respect to different numbers of concurrently issued TPC-H Q6 queries as a measure of communication costs. As shown in Figure 6, when compared to [10] and [95], Engorgio can reduce the query and response sizes by up to 1,014× and 976×, respectively.

Lastly, when all the 48 processor cores are utilized, the amortized runtime of Engorgio can be under 3 and 75 seconds for TPC-H Q1 and HQ2, respectively, over a 10K-row EDB.

Remark: Although Engorgio provides substantial latency reductions for the queries over hybrid EDBs, we still identify the homomorphic comparison function as the fundamental challenge against FHE-based EDBs. In particular, the current approximation polynomials in implementing the homomorphic comparison function yet face the limited precision and high multiplicative depth problems. Thus, finding better polynomials to approximate the comparison function is one of the key future research directions.

7 Conclusions

In this work, we present Engorgio, an FHE-based encrypted hybrid database framework that evaluates arbitrary-precision hybrid SQL statements on large-size databases. By designing a set of new data ordering and querying primitives, we are able to efficiently apply both relational and vectorized queries over arbitrary-precision data. In the experiments, we show that Engorgio can outperform the SOTA FHE-based EDB frameworks across a comprehensive set of SQL benchmarks.

Acknowledgement

We thank anonymous reviewers and shepherds for their helpful feedback. This work was supported by the National Key R&D Program of China (2023YFB3106200) and the National Natural Science Foundation of China (U21B2021, 61932014, 62472015, T2425023, 624B2016). This work was also supported by Huawei Technologies Co., Ltd. and Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing.

Ethics Considerations And Compliance With The Open Science Policy

This section outlines the ethical considerations and compliance with the open science policy of this work.

Ethics Considerations

This work proposes an encrypted hybrid database framework, which mainly focuses on the protection of sensitive data in hybrid databases. Importantly, the experimental portion of this research does not involve live systems but is conducted in a controlled environment. Below, we discuss the ethical implications of the research, adherence to open science principles, and our compliance with the relevant guidelines.

• **Beneficence**: The main objective of our research is to protect data security, which aligns with the idea of beneficence. By developing and evaluating Engorgio, we seek to enhance data security and prevent unauthorized access, thereby protecting users from potential harms associated with data breaches. Based on our database outsourcing research background, we have conducted a thorough risk assessment and ensured that our methodologies will not produce potential negative outcomes. The experimental setup is contained within a secure and isolated environment to avoid any negative impacts on public systems or individuals.

• **Respect For Persons**: Respect for the individual is one of the core principles of this work. This is because the original intention of this work is to protect the security of personal sensitive data in an outsourced environment while ensuring the security of sensitive data in query statements. We maintain high standards of integrity in all aspects of the research process. All data used in the experiments are anonymized and securely managed to ensure privacy and confidentiality.

• **Justice**: The principle of justice requires a fair distribution of the benefits and burdens of research. In our work, we ensure that users of the outsourced database benefit from the server's high computational resources while maintaining data security. We also make sure that the server does not result in data breaches or other issues, thus enhancing service quality and ensuring that both parties involved benefit from the research. Additionally, our research does not use sensitive or personal data from public systems, avoiding any unfair negative impacts on individuals or groups.

• **Respect for Law and Public Interest**: Our work follows all pertinent laws and guidelines on data security and cryptographic research. We have also looked over our procedures to make sure Engorgio follows public interest criteria and does not support any illegal behavior or unethical activity.

Open Science Policy

Following the open science policy, we are dedicated to guaranteeing that our work is transparent and easily available. The following actions are in place:

• **Reproducibility**: To facilitate reproducibility and independent validation of our work, we will provide detailed descriptions of our experimental setup and results. Additionally, all relevant research artifacts will be submitted for the artifact evaluation process. This submission will ensure the availability, functionality, and reproducibility of our work, allowing for thorough assessment by the committee.

• **Open Access**: Our source code and materials for replicating experiments are publicly available at https://zenodo.org/records/14730651.

References

[1] ADDANKI, S., GARBE, K., JAFFE, E., AND RAFAIL OSTROVSKY, A. P. Prio+: Privacy preserving aggregate statistics via boolean shares. In *SCN* (2022), vol. 13409, pp. 516–539.

[2] ALBRECHT, M. R., PLAYER, R., AND SCOTT, S. On the concrete hardness of learning with errors. *J. Math. Cryptol.* (2015), 169–203.

[3] ARAKI, T., BARAK, A., FURUKAWA, J., ET AL. Optimized honest-majority mpc for malicious adversaries—breaking the 1 billion-gate per second barrier. In *IEEE SP* (2017), pp. 843–862.

[4] AWS. Machine Learning on AWS. https://aws.amazon.com/machine-learning/?nc2=h_ql_sol_use_ml, 2023. Accessed: 2023-01-01.

[5] AZURE. Azure Machine Learning. https://azure.microsoft.com/en-us/products/machine-learning/, 2023. Accessed: 2023-01-01.

[6] BADAWI, A. A., ALEXANDRU, A., BATES, J., ET AL. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022.

[7] BAE, Y., CHEON, J. H., CHO, W., ET AL. META-BTS: Bootstrapping Precision Beyond the Limit. In *ACM CCS* (2022), pp. 223–234.

[8] BATCHER, K. E. Sorting networks and their applications. In *American Federation of Information Processing Societies* (1968), pp. 307–314.

[9] BATER, J., ELLIOTT, G., EGGEN, C., GOEL, S., KHO, A., AND ROGERS, J. Smcql: Secure querying for federated databases. *arXiv preprint arXiv:1606.06808* (2016).

[10] BIAN, S., ZHANG, Z., PAN, H., MAO, R., ZHAO, Z., JIN, Y., AND GUAN, Z. HE3DB: an efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption. In *ACM CCS* (2023), pp. 2930–2944.

[11] BOLDYREVA, A., AND TANG, T. Privacy-preserving approximate k-nearest-neighbors search that hides access, query and volume patterns. *Proc. Priv. Enhancing Technol. 2021* (2021), 549–574.

[12] BOSSUAT, J., MOUCHET, C., TRONCOSO-PASTORIZA, J. R., AND HUBAUX, J. Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys. In *EUROCRYPT* (2021), vol. 12696, pp. 587–617.

[13] BOSSUAT, J.-P., CAMMAROTA, R., CHEON, J. H., ET AL. Security Guidelines for Implementing Homomorphic Encryption. Cryptology ePrint Archive, Paper 2024/463, 2024.

[14] BOYLE, E., CHANDRAN, N., GILBOA, N., ET AL. Function secret sharing for mixed-mode and fixed-point secure computation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2021), pp. 871–900.

[15] BOYLE, E., GILBOA, N., AND ISHAI, Y. Secure computation with preprocessing via function secret sharing. In *Theory of Cryptography International Conference, TCC* (2019), vol. 11891, pp. 341–371.

[16] BRAKERSKI, Z. Fully homomorphic encryption without modulus switching from classical gapsvp. In *CRYPTO* (2012), pp. 868–886.

[17] BROUCKE, R. Ten subroutines for the manipulation of chebyshev series [C1] (algorithm 446). *Commun. ACM 16* (1973), 254–256.

[18] CADWALLADR, C., AND GRAHAM-HARRISON, E. Revealed: 50 million facebook profiles harvested for cambridge analytica in major data breach. *The guardian 17* (2018), 22.

[19] CHATTERJEE, A., KAUSHAL, M., AND SENGUPTA, I. Accelerating sorting of fully homomorphic encrypted data. In *INDOCRYPT 2013* (2013), vol. 8250, pp. 262–273.

[20] CHAZAN, D. Introduction to Approximation Theory (E. W. Cheney). *SIAM Review 10*, 3 (1968), 393–394.

[21] CHEN, H., CHILLOTTI, I., DONG, Y., POBURINNAYA, O., RAZENSHTEYN, I. P., AND RIAZI, M. S. SANNS: scaling up secure approximate k-nearest neighbors search. In *USENIX Security Symposium* (2020), pp. 2111–2128.

[22] CHEN, H., CHILLOTTI, I., AND REN, L. Onion ring ORAM: efficient constant bandwidth oblivious RAM from (leveled) TFHE. In *ACM CCS* (2019), pp. 345–360.

[23] CHENG, Y., WANG, D., ZHOU, P., AND ZHANG, T. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine 35* (2018), 126–136.

[24] CHENG, Y., WANG, D., ZHOU, P., AND ZHANG, T. A survey of model compression and acceleration for deep neural networks, 2020.

[25] CHEON, J. H., HAN, K., KIM, A., KIM, M., AND SONG, Y. A Full RNS Variant of Approximate Homomorphic Encryption. Cryptology ePrint Archive, Paper 2018/931, 2018.

[26] CHEON, J. H., HAN, K., KIM, A., KIM, M., AND SONG, Y. Bootstrapping for Approximate Homomorphic Encryption. In *EUROCRYPT* (2018), vol. 10820, pp. 360–384.

[27] CHEON, J. H., KIM, D., AND KIM, D. Efficient homomorphic comparison methods with optimal complexity. In *ASIACRYPT* (2020), pp. 221–256.

[28] CHEON, J. H., KIM, D., KIM, D., LEE, H., AND LEE, K. Numerical method for comparison on homomorphically encrypted numbers. In *ASIACRYPT* (2019), pp. 415–445.

[29] CHEON, J. H., SON, Y., AND YHEE, D. Practical FHE parameters against lattice attacks. *IACR Cryptol. ePrint Arch.* (2021), 39.

[30] CLOUD, G. Cloud SQL. https://cloud.google.com/sql/, 2023. Accessed: 2023-01-01.

[31] CONG, K., DAS, D., PARK, J., AND PEREIRA, H. V. L. SortingHat: Efficient Private Decision Tree Evaluation via Homomorphic Encryption and Transciphering. In *ACM CCS* (2022), pp. 563–577.

[32] CONG, K., GEELEN, R., KANG, J., AND PARK, J. Revisiting Oblivious Top-k Selection with Applications to Secure k-NN Classification. Cryptology ePrint Archive, Paper 2023/852, 2023.

[33] CORRIGAN-GIBBS, H., AND BONEH, D. Prio: Private, Robust, and Scalable Computation of Aggregate Statistics. In *USENIX Conference on Networked Systems Design and Implementation* (2017), p. 259–282.

[34] COUNCIL, T. P. P. TPC BENCHMARK^{TM} H Standard Specification. Tech. rep., 2022.

[35] CUI, N., YANG, X., WANG, B., LI, J., AND WANG, G. Svknn: Efficient secure and verifiable k-nearest neighbor query on the cloud platform*. In *IEEE International Conference on Data Engineering* (2020), pp. 253–264.

[36] DALSKOV, A. P. K., ESCUDERO, D., AND KELLER, M. Secure evaluation of quantized neural networks. *Proc. Priv. Enhancing Technol. 2020* (2020), 355–375.

[37] DAUTERMAN, E., RATHEE, M., POPA, R. A., AND STOICA, I. Waldo: A Private Time-Series Database from Function Secret Sharing. In *IEEE SP* (2022), pp. 2450–2468.

[38] DEMERTZIS, I., PAPADOPOULOS, D., PAPAMANTHOU, C., AND SHINTRE, S. Seal: Attack mitigation for encrypted databases via adjustable leakage. In *USENIX Security Symposium* (2020), pp. 2433–2450.

[39] DEVADAS, S., VAN DIJK, M., FLETCHER, C. W., REN, L., SHI, E., AND WICHS, D. Onion oram: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography Conference* (2016), pp. 145–174.

[40] ELMEHDWI, Y., SAMANTHULA, B. K., AND JIANG, W. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *IEEE International Conference on Data Engineering* (2014), pp. 664–675.

[41] ESKANDARIAN, S., AND ZAHARIA, M. ObliDB: Oblivious Query Processing for Secure Databases. *Proc. VLDB Endow. 13*, 2 (2019), 169–183.

[42] GUIMARÃES, A., BORIN, E., AND ARANHA, D. F. Revisiting the functional bootstrap in TFHE. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2 (2021), 229–253.

[43] HACKENJOS, T., HAHN, F., AND KERSCHBAUM, F. SAGMA: Secure Aggregation Grouped by Multiple Attributes. In *SIGMOD* (2020), pp. 587–601.

[44] HALEVI, S., POLYAKOV, Y., AND SHOUP, V. An improved RNS variant of the BFV homomorphic encryption scheme. In *CT-RSA* (2019), pp. 83–105.

[45] HALEVI, S., AND SHOUP, V. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.* (2020).

[46] HAN, K., HHAN, M., AND CHEON, J. H. Improved homomorphic discrete fourier transforms and FHE bootstrapping. *IEEE Access 7* (2019), 57361–57370.

[47] HENZINGER, A., DAUTERMAN, E., CORRIGAN-GIBBS, H., AND ZELDOVICH, N. Private Web Search with Tiptoe. In *SOSP* (2023), p. 396–416.

[48] HONG, S., KIM, S., CHOI, J., LEE, Y., AND CHEON, J. H. Efficient sorting of homomorphic encrypted data with k-way sorting network. *IEEE Trans. Inf. Forensics Secur. 16* (2021), 4389–4404.

[49] ILIASHENKO, I., AND ZUCCA, V. Faster homomorphic comparison operations for BGV and BFV. *Proc. Priv. Enhancing Technol. 2021* (2021), 246–264.

[50] ILYAS, I. F., BESKALES, G., AND SOLIMAN, M. A. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv. 40* (2008), 11:1–11:58.

[51] ISHAI, Y., KUSHILEVITZ, E., LU, S., AND OSTROVSKY, R. Private large-scale databases with distributed searchable symmetric encryption. In *Cryptographers' Track at the RSA Conference* (2016), pp. 90–107.

[52] JÉGOU, H., DOUZE, M., AND SCHMID, C. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2011).

[53] KELLARIS, G., KOLLIOS, G., NISSIM, K., AND O'NEILL, A. Generic attacks on secure outsourced databases. In *ACM CCS* (2016), pp. 1329–1340.

[54] KELLER, M., AND SUN, K. Secure quantized training for deep learning. In *ICML 2022*, vol. 162, pp. 10912–10938.

[55] KORTEKAAS, Y. Access Pattern Hiding Aggregation over Encrypted Databases, October 2020.

[56] KUMAWAT, S., AND NAGAHARA, H. Privacy-preserving action recognition via motion difference quantization. In *ECCV 2022*, vol. 13673, pp. 518–534.

[57] LANG, N., SOFER, E., SHAKED, T., AND SHLEZINGER, N. Joint privacy enhancement and quantization in federated learning. *IEEE Trans. Signal Process. 71* (2023), 295–310.

[58] LEE, E., LEE, J., NO, J., AND KIM, Y. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE TDSC 19* (2022), 3711–3727.

[59] LEE, Y., LEE, J., KIM, Y., KIM, Y., NO, J., AND KANG, H. High-Precision Bootstrapping for Approximate Homomorphic Encryption by Error Variance Minimization. In *EUROCRYPT*, vol. 13275, pp. 551–580.

[60] LEWIS, P., PEREZ, E., PIKTUS, A., ET AL. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems 33*.

[61] LI, J., HUANG, Z., ZHANG, M., LIU, J., HONG, C., WEI, T., AND CHEN, W. PANTHER: Private approximate nearest neighbor search in the single server setting. Cryptology ePrint Archive, Paper 2024/1774, 2024.

[62] LIAGOURIS, J., KALAVRI, V., FAISAL, M., AND VARIA, M. Secrecy: Secure collaborative analytics on secret-shared data. *arXiv preprint arXiv:2102.01048* (2021).

[63] LOPES, C. C., TIMES, V. C., MATWIN, S., CIFERRI, R. R., AND DE AGUIAR CIFERRI, C. D. Processing OLAP queries over an en-

crypted data warehouse stored in the cloud. In *Data Warehousing and Knowledge Discovery* (2014), vol. 8646, pp. 195–207.

[64] LU, W., HUANG, Z., HONG, C., MA, Y., AND QU, H. PEGASUS: Bridging Polynomial and Non-polynomial Evaluations in Homomorphic Encryption. In *IEEE SP* (2021), pp. 1057–1073.

[65] LUO, Y., XU, N., PENG, H., WANG, C., DUAN, S., MAHMOOD, K., WEN, W., DING, C., AND XU, X. AQ2PNN: enabling two-party privacy-preserving deep neural network inference with adaptive quantization. In *MICRO 2023*, pp. 628–640.

[66] LYUBASHEVSKY, V., PEIKERT, C., AND REGEV, O. On ideal lattices and learning with errors over rings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2010), pp. 1–23.

[67] MAYANS, R. The chebyshev equioscillation theorem. *Journal of Online Mathematics and Its Applications 6* (2006).

[68] MONGODB. Application-driven analytics. https://www.mongodb.com/use-cases/analytics, 2023. Accessed: 2023-01-01.

[69] MUGHEES, M. H., AND REN, L. Vectorized Batch Private Information Retrieval. Cryptology ePrint Archive, Paper 2022/1262, 2022.

[70] OPENFHEORG. OpenFHE - Open-Source Fully Homomorphic Encryption Library.

[71] PAPADIMITRIOU, A., BHAGWAN, R., CHANDRAN, N., ET AL. Big Data Analytics over Encrypted Datasets with Seabed. In *USENIX Conference on Operating Systems Design and Implementation* (2016), p. 587–602.

[72] PODDAR, R., BOELTER, T., AND POPA, R. A. Arx: An Encrypted Database using Semantically Secure Encryption. *Proc. VLDB Endow. 12*, 11.

[73] PODDAR, R., KALRA, S., YANAI, A., DENG, R., POPA, R. A., AND HELLERSTEIN, J. M. Senate: A maliciously-secure mpc platform for collaborative analytics. In *30th USENIX Security Symposium* (2021), pp. 2129–2146.

[74] POPA, R. A., REDFIELD, C. M. S., ZELDOVICH, N., AND BALAKRISHNAN, H. CryptDB: protecting confidentiality with encrypted query processing. In *ACM SOSP* (2011), pp. 85–100.

[75] PRIEBE, C., VASWANI, K., AND COSTA, M. Enclavedb: A secure database using SGX. In *IEEE SP* (2018), pp. 264–278.

[76] REGEV, O. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM 56*, 6 (2009), 34.

[77] REN, X., SU, L., GU, Z., WANG, S., LI, F., XIE, Y., BIAN, S., LI, C., AND ZHANG, F. Heda: Multi-attribute unbounded aggregation over homomorphically encrypted database. *Proc. VLDB Endow. 16*, 4 (2022), 601–614.

[78] RIVLIN, T. The chebyshev polynomials, pure and applied mathematics, 1974.

[79] SERVAN-SCHREIBER, S., LANGOWSKI, S., AND DEVADAS, S. Private approximate nearest neighbor search with sublinear communication. In *IEEE SP* (2022), pp. 911–929.

[80] SHANBHAG, A., PIRK, H., AND MADDEN, S. Efficient Top-K Query Processing on Massively Parallel Hardware. In *SIGMOD* (2018).

[81] SHAUL, H., FELDMAN, D., AND RUS, D. Secure k-ish nearest neighbors classifier. *Proc. Priv. Enhancing Technol.* (2020), 42–61.

[82] SRIVASTAVA, S., AND SINGH, A. K. Fraud detection in the distributed graph database. *Clust. Comput. 26*, 1 (2023), 515–537.

[83] SUCHAL, J., AND NÁVRAT, P. Full text search engine as scalable k-nearest neighbor recommendation system. In *IFIP Artificial Intelligence in Theory and Practice* (2010), pp. 165–173.

[84] TRAUTMAN, L. J., AND ORMEROD, P. C. Corporate directors' and officers' cybersecurity standard of care: The yahoo data breach. *American University Law Review 66*, 5 (2017), 3.

[85] TU, S., ZHENG, W., KOHLER, E., LISKOV, B., AND MADDEN, S. Speedy transactions in multicore in-memory databases. In *SOSP* (2013), p. 18–32.

[86] VOLGUSHEV, N., SCHWARZKOPF, M., GETCHELL, B., VARIA, M., LAPETS, A., AND BESTAVROS, A. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019* (2019), pp. 1–18.

[87] WANG, B., HOU, Y., AND LI, M. Practical and secure nearest neighbor search on encrypted large-scale data. In *IEEE International Conference on Computer Communications* (2016), pp. 1–9.

[88] WANG, C., BATER, J., NAYAK, K., AND MACHANAVAJJHALA, A. Inc-Shrink: Architecting Efficient Outsourced Databases using Incremental MPC and Differential Privacy. In *SIGMOD* (2022), pp. 818–832.

[89] WANG, J., YI, X., GUO, R., ET AL. Milvus: A purpose-built vector data management system. In *SIGMOD* (2021), pp. 2614–2627.

[90] WANG, Y., AND YI, K. Secure yannakakis: Join-aggregate queries over private data. In *SIGMOD* (2021), pp. 1969–1981.

[91] WEI, C., WU, B., WANG, S., LOU, R., ZHAN, C., LI, F., AND CAI, Y. Analyticdb-v: A hybrid analytical engine towards query fusion for structured and unstructured data. *Proc. VLDB Endow.* (2020), 3152–3165.

[92] YAO, B., LI, F., AND XIAO, X. Secure nearest neighbor revisited. In *IEEE International Conference on Data Engineering* (2013), pp. 733–744.

[93] ZAMA. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.

[94] ZHANG, Q., XU, S., CHEN, Q., ET AL. VBASE: unifying online vector similarity search and relational queries via relaxed monotonicity. In *USENIX Symposium on Operating Systems Design and Implementation* (2023), pp. 377–395.

[95] ZHANG, Z., BIAN, S., ZHAO, Z., MAO, R., ZHOU, H., HUA, J., JIN, Y., AND GUAN, Z. ArcEDB: An Arbitrary-Precision Encrypted Database via (Amortized) Modular Homomorphic Encryption. In *ACM CCS* (2024), p. 4613–4627.

[96] ZHENG, W., TU, S., KOHLER, E., AND LISKOV, B. Fast Databases with Fast Durability and Recovery Through Multicore Parallelism. In *OSDI* (2014).

[97] ZHENG, Y., LU, R., GUAN, Y., SHAO, J., AND ZHU, H. Efficient and privacy-preserving similarity range query over encrypted time series data. *IEEE TDSC 19*, 4 (2022), 2501–2516.

[98] ZHU, H., WANG, R., JIN, Y., LIANG, K., AND NING, J. Distributed additive encryption and quantization for privacy preserving federated deep learning. *Neurocomputing 463* (2021), 309–327.

[99] ZUBER, M., AND SIRDEY, R. Efficient homomorphic evaluation of k-nn classifiers. *Proc. Priv. Enhancing Technol. 2021*, 2 (2021), 111–129.

A SECURITY PROOF

We assume that the client performs the encryption function over the hybrid database $\mathcal{D}$ and the query $\mathcal{Q}$ locally. The server carries out the homomorphic query evaluation over the encrypted hybrid database $[\mathcal{D}]$ using the encrypted query $[\mathcal{Q}]$. This process yields the encrypted query result $[\mathcal{R}]$ along with certain intermediate results (e.g., $[\mathcal{F}]$, $[\mathsf{D}]$, $[\mathcal{O}]$). From the encryption procedures mentioned in Section 2.1, we know that the client holds the private key $\tilde{s}$. We safely hold it true that the original RNS-CKKS cryptosystem [25] attains indistinguishability under chosen-plaintext attack (i.e., achieves IND-CPA security). In addition, we assume that the adversary $\mathcal{A}$ (the server) is semi-honest.

We prove Engorgio's security guarantees under the simulation paradigm by defining two worlds: the real world and the ideal world. In the real world, the actual protocol is run by honest parties. In contrast, in the ideal world, an ideal functionality $\mathfrak{F}$ takes inputs from the parties and directly outputs the result to the relevant party. We construct a simple simulator $\mathcal{S}$ that simulates the view of the semi-honest server $\mathcal{A}$ on our system. Basically, $\mathcal{S}$ simulates the behavior of the

client within the homomorphic query evaluation protocol π, all while remaining unaware of any input. Recall that in the Engorgio protocol, the server carries out the homomorphic query evaluation over the encrypted hybrid database $[\mathcal{D}]$ using the encrypted query $[Q]$. This process yields the encrypted query result $[\mathcal{R}]$ along with certain intermediate results (e.g., $[\mathcal{F}]$, [D], $[\mathcal{O}]$). While querying operations in Engorgio (i.e., `HomFilter`, `HomDist`, `HomOrder`, `HomAgg`, and `HomGROUP`) can be expressed as the combination of basic homomorphic operations (i.e., multiplication, addition, rotation, etc.), we collectively refer to these homomorphic query operations as `HQO` and the plain query operations as `PQO`.

Here, We first prove that the view of any adversary interacting with the homomorphic query operation `HQO` in Engorgio is indistinguishable from their view of the original RNS-CKKS cryptosystem, as detailed in Section 2.1. Given that the intermediate results of the combination of multiple `HQO`s are still valid FHE ciphertexts, and there is no intermediate decryption operation in Engorgio, we further prove that the combination of multiple `HQO`s against semi-honest adversaries. Finally, since the Engorgio protocol is a single-round protocol, there is only one round of interaction during the query process: the user sends the encrypted query, and the server sends back the encrypted query result. No additional communication or decryption operations occur in between, ensuring that the protocol does not leak any information under the security parameter of λ. Thus we conclude that Engorgio securely evaluates the ideal functionality $\mathfrak{F}$ when instantiated with semantically secure homomorphic encryption, and is secure against semi-honest adversaries with a computational security parameter of λ. Due to space limitation, we refer to the full version of the manuscript for the complete proof.

B DETAIL ALGORITHMS

B.1 Permutation Matrix Encoding

As shown in Algorithm 6, given the comparison result $ct_{\mathtt{cmp}_{s,p}}$, `MEncode` apply `Rotate` to $ct_{\mathtt{cmp}_{s,p}}$ and add the rotate result with $ct_{\mathtt{cmp}_{s,p}}$ to obtain the $ct_{\mathtt{cmp}_{\mathrm{diag}}}$ on Line 1. Then on Line 2-3 apply `HomGate` and `Rotate` to obtain $ct_{\mathtt{cmp}_{\mathrm{sup}}}$ and $ct_{\mathtt{cmp}_{\mathrm{sub}}}$, thus the permutation matrix is encoded as $\mathbf{M}_{s,p} = [ct_{\mathtt{cmp}_{\mathrm{sup}}}, ct_{\mathtt{cmp}_{\mathrm{diag}}}, ct_{\mathtt{cmp}_{\mathrm{sub}}}]$.

B.2 Homomorphic Synchronization

As detailed in Algorithm 7, given an unsorted QRLWE ciphertext, and sorting matrices, we synchronize this ciphertext base on the sorting matrices by performing `HomOrdApp` on Line 4. Therefore output a QRLWE ciphertext ct_{res}, which encrypts a synchronized column.

Algorithm 6: Matrix Encoding `MEncode`

Input : Comparison result $ct_{\mathtt{cmp}_{s,p}}$, rotate key $\mathbf{RK}$.
Input : Table size $|\mathcal{T}|_{\mathrm{row}}$, stage s, part p.
Output : Encoded permutation Matrix $\mathbf{M}_{s,p}$.

1 $ct_{\mathtt{cmp}_{\mathrm{diag}}} \leftarrow ct_{\mathtt{cmp}_{s,p}} + \mathtt{Rotate}(ct_{\mathtt{cmp}_{s,p}}, \mathbf{RK}, |\mathcal{T}|_{\mathrm{row}} - 2^{s-p})$
2 $ct_{\mathtt{cmp}_{\mathrm{sub}}} \leftarrow \mathtt{HomGate}(ct_{\mathtt{cmp}_{s,p}}, \mathtt{NOT})$
3 $ct_{\mathtt{cmp}_{\mathrm{sup}}} \leftarrow \mathtt{Rotate}(ct_{\mathtt{cmp}_{\mathrm{sub}}}, \mathbf{RK}, |\mathcal{T}|_{\mathrm{row}} - 2^{s-p})$

Return : $\mathbf{M}_{s,p} = [ct_{\mathtt{cmp}_{\mathrm{sup}}}, ct_{\mathtt{cmp}_{\mathrm{diag}}}, ct_{\mathtt{cmp}_{\mathrm{sub}}}]$

Algorithm 7: Homomorphic Synchronization `HomSync`

Input : An unsorted QRLWE ciphertext $ct_{0,0}$. Sorting matrices $[\mathbf{M}_{0,0}, \mathbf{M}_{1,0}, \cdots, \mathbf{M}_{\log_2 N-1, \log_2 N}]$, rotate key $\mathbf{RK}$, and precision parameter (Θ, α).
Output : A QRLWE ciphertext ct_{res}, which encrypts a synchronized column.

1 $ct_{\mathrm{res}} \leftarrow ct_{0,0}$
2 **for** $s = 0$ **to** $\log_2 |\mathcal{T}|_{\mathrm{row}}$ **do**
3 **for** $p = 0$ **to** s **do**
4 $\mathtt{HomOrdApp}(ct_{\mathrm{res}}, \mathbf{M}_{s,p}, |\mathcal{T}|_{\mathrm{row}}, s, p, \mathbf{RK}, (\Theta, \alpha))$

Return : ct_{res}

B.3 Unbounded-Size Sorting

The core idea of unbounded-size sorting is to treat multiple ciphertexts that need to be sorted as a whole ciphertext, arrange the multiple permutation matrices along the diagonal into a large permutation matrix, and then execute the sorting algorithm. We first propose `CrossRot` to perform homomorphic rotation over multiple ciphertexts. As detailed in Algorithm 8, for a QRLWE ciphertext vector $\mathbf{CT}$ and rotate step T, we divide the rotation into two parts, the giant part $\lfloor \frac{T}{N} \rfloor$ and the baby part $T \bmod N$. For the giant part, we rearrange the positions of each QRLWE ciphertext (Line 1-3). For the baby part, we rotate the adjacent ciphertexts, multiply them by the mask vectors corresponding to the adjacent ciphertexts, and merge them together (Line 5-11). We use `CrossRot` to replace `Rotate` in Algorithm 3, `MEncode`, and Algorithm 4, resulting in `UHomOrdGen`, `UMatEncode`, and `UHomOrdApp` for unbounded-size sorting algorithm Algorithm 9. The Algorithm 9 first initialize mask vector and the eliminate vector similar to Algorithm 3, but here the vector length is Nl (Line 1-9). Then we use `CrossRot` and `ArbQuantComp` to carry out pairwise comparisons of different positions in ciphertext vector according to the sorting network (Line 10-13). Then we invoke `UMatEncode` to encode the permutation matrix (Line 14). Finally, we invoke 2 `CrossRot` and $3l$ homomorphic multiplications for each ciphertext segment to perform permutation (Line 15-20).

Algorithm 8: Cross-Ciphertext Rotation `CrossRot`

Input : A RLWE ciphertext vector $\mathbf{CT}$, which contains l RLWE ciphertexts, encrypts Nl values, rotate key $\mathbf{RK}$ and rotate step T.

Output : A RLWE ciphertext vector $\mathbf{CT}_{\text{rot}}$, which contains l RLWE ciphertexts after cyclic left shift T position.

1 $giant \leftarrow \lfloor \frac{T}{N} \rfloor$
2 **for** $i = 0$ **to** $l-1$ **do**
3 $\quad \mathbf{CT}_{\text{prerot}}[i] \leftarrow \mathbf{CT}[(i+giant) \mod l]$
4 $baby \leftarrow T \mod N$
5 $\mathbf{Mask}_0[0 : N-baby] \leftarrow \{1\}^{N-baby}$
6 $\mathbf{Mask}_0[N-baby : N] \leftarrow \{0\}^{baby}$
7 $\mathbf{Mask}_1 \leftarrow \{1\}^N - \mathbf{Mask}_0$
8 **for** $j = 0$ **to** $l-2$ **do**
9 $\quad \mathbf{CT}_{\text{rot}}[j] \leftarrow$ `Rotate`$(\mathbf{CT}_{\text{prerot}}[j], \mathbf{RK}, baby) \cdot \mathbf{Mask}_0 +$ `Rotate`$(\mathbf{CT}_{\text{prerot}}[j+1], \mathbf{RK}, baby) \cdot \mathbf{Mask}_1$
10 $\mathbf{CT}_{rot}[l-1] \leftarrow$ `Rotate`$(\mathbf{CT}_{\text{prerot}}[l], \mathbf{RK}, baby) \cdot \mathbf{Mask}_0 +$ `Rotate`$(\mathbf{CT}_{\text{prerot}}[0], \mathbf{RK}, baby) \cdot \mathbf{Mask}_1$

Return : $\mathbf{CT}_{\text{rot}}$

B.4 Batched Sorting

As detailed in Algorithm 10, given an unsorted QRLWE ciphertext $ct_{0,0}$, which encrypts $N/|\mathcal{T}|_{\text{row}}$ unsorted columns, table size $|\mathcal{T}|_{\text{row}}$ (padding to a power of two), rotate key $\mathbf{RK}$, and precision parameter (Θ, α). `BatchedSort` outputs QRLWE ciphertext ct_{res}, which encrypts $N/|\mathcal{T}|_{\text{row}}$ sorted columns. We iteratively perform the aforementioned `HomOrdGen`, `MEncode`, and `HomOrdApp` on Line 1-6, but terminate the sorting process early in stage $s = \log_2 |\mathcal{T}|_{\text{row}}$to obtain the $N/|\mathcal{T}|_{\text{row}}$ sorted columns. For simplicity, we assume that there is only one ciphertext after packing, that is, $\frac{x|\mathcal{T}|_{\text{row}}}{N} = 1$. When $\frac{x|\mathcal{T}|_{\text{row}}}{N} > 1$, we use unbounded-sort proposed in Section 5.1 to execute the above steps.

B.5 Homomorphic Top-k

As detailed in Algorithm 11, given an unsorted QRLWE ciphertext $ct_{0,0}$, rotate key $\mathbf{RK}$, precision parameter (Θ, α), and Top-k parameter k, We first perform the first $\log_2(k)$ stages of the sorting algorithm on Line 1-6, and then execute the $\log_2(k)$-th stage for $\log_2(N) - \log_2(k) - 1$ times on Line 7-11. During t-th execution of the $\log_2(k)$-th stage, we perform $part_0$ on Line 7-22 by adjusting the rotation step of $part_0$ in `MEncode`, `HomOrdGen`, and `HomOrdApp` to $2^{\log_2(k)} + (2^{t+1} - 1) \cdot 2k$. Finally, `HomTopK` outputs QRLWE ciphertext ct_{res}, where the first k elements are the top-k results.

Algorithm 9: Unbounded Homomorphic Sort `UnboundedSort`

Input : An unsorted QRLWE ciphertext vector $\mathbf{CT}_{0,0}$, which contains l QRLWE ciphertexts, each ciphertext encrypts a plaintext with length N.

Input : Table size Nl and rotate key $\mathbf{RK}$.

Output : A QRLWE ciphertext vector $\mathbf{CT}_{\text{res}}$, which encrypts a sorted vector result.

1 $\mathbf{CT}_{\text{res}} \leftarrow \mathbf{CT}_{0,0}$
2 **for** $s = 0$ **to** $\log_2(Nl) - 1$ **do**
3 $\quad$ **for** $p = 0$ **to** s **do**
4 $\quad\quad$ **for** $i = 0$ **to** Nl **step** 2^{s+2} **do**
5 $\quad\quad\quad \mathbf{Mask}[i : i + 2^{s+1}] \leftarrow 1$
6 $\quad\quad\quad \mathbf{Mask}[i + 2^{s+1} : i + 2^{s+2} - 1] \leftarrow -1$
7 $\quad\quad$ **for** $i = 0$ **to** Nl **step** 2^{s-p+1} **do**
8 $\quad\quad\quad \mathbf{Eli}[i : i + 2^{s-p}] \leftarrow 1$
9 $\quad\quad\quad \mathbf{Eli}[i + 2^{s-p} : i + 2^{s-p+1} - 1] \leftarrow 0$
10 $\quad\quad$ **for** $i = 0$ **to** $w-1$ **do**
11 $\quad\quad\quad \mathbf{CT}_{\text{rot}}[i] \leftarrow$ `CrossRot`$(\mathbf{CT}_{s,p}[i] \cdot \mathbf{Mask}, \mathbf{RK}, 2^{s-p}) \cdot \mathbf{Eli}$
12 $\quad\quad\quad \mathbf{CT}_{\text{mask}}[i] \leftarrow \mathbf{CT}_{s,p}[i] \cdot \mathbf{Mask} \cdot \mathbf{Eli}$
13 $\quad\quad \mathbf{CT}_{\text{cmp}_{s,p}}[i] \leftarrow$ `ArbQuantComp`$(\mathbf{CT}_{\text{mask}}[i], \mathbf{CT}_{\text{rot}}[i], >, (\Theta, \alpha))$
14 $\quad\quad \mathbf{M}_{s,p} \leftarrow$ `MEncode`$(ct_{\text{cmp}_{s,p}}, \mathbf{RK}, N, s, p)$
15 $\quad\quad$ **for** $i = 0$ **to** w **do**
16 $\quad\quad\quad \mathbf{CT}_{\text{sup}}[i] \leftarrow$ `CrossRot`$(\mathbf{CT}_{s,p}[i], \mathbf{RK}, N - 2^{s-p})$
17 $\quad\quad\quad \mathbf{CT}_{\text{sub}}[i] \leftarrow$ `CrossRot`$(\mathbf{CT}_{s,p}[i], \mathbf{RK}, 2^{s-p})$
18 $\quad\quad$ **for** $i = 0$ **to** $w-1$ **do**
19 $\quad\quad\quad$ **for** $j = 0$ **to** $l-1$ **do**
20 $\quad\quad\quad\quad \mathbf{CT}_{\text{res}}[i][j] \leftarrow \mathbf{CT}_{\text{sup}}[i][j] \cdot \mathbf{M}_{s,p}[0][i] + \mathbf{CT}_{s,p}[i][j] \cdot \mathbf{M}_{s,p}[1][i] + \mathbf{CT}_{\text{sub}}[i][j] \cdot \mathbf{M}_{s,p}[2][i]$

$\quad$ **Return :** $\mathbf{CT}_{\text{res}}$

C EXPERIMENT DETAILS

C.1 Microbenchmark Details

In the filtering test, we divide ArcEDB [95] comparison into the offline phase and the online phase. This is because the comparison algorithm of [95] requires an RLWE ciphertext and an RGSW ciphertext as input. The offline phase involves the direct comparison of RGSW ciphertexts generated by the user. In contrast, the online phase is required for more complex operations, such as inner table comparisons or comparisons after filtering or aggregation. These operations necessitate a sophisticated ciphertext conversion algorithm to transform RLWE ciphertexts into RGSW ciphertexts. For the synchronization benchmark, since both [95] and [22] require RGSW ciphertext as input, we add ciphertext conversion

Algorithm 10: Batched Sort `BatchedSort`

Input : An unsorted QRLWE ciphertext $ct_{0,0}$, which encrypts $N/|\mathcal{T}|_{\text{row}}$ unsorted columns.
Input : Column length $|\mathcal{T}|_{\text{row}}$, rotate key **RK**, and precision parameter (Θ, α).
Output : A QRLWE ciphertext ct_{res}, which encrypts encrypts $N/|\mathcal{T}|_{\text{row}}$ sorted columns.

$ct_{\text{res}} \leftarrow ct_{0,0}$
for $s = 0$ **to** $\log_2(|\mathcal{T}|_{\text{row}})$ **do**
  for $p = 0$ **to** s **do**
    $ct_{\text{cmp}_{s,p}} \leftarrow \texttt{HomOrdGen}(ct_{\text{res}}, |\mathcal{T}|_{\text{row}}, s, p, \mathbf{RK}, (\Theta, \alpha))$
    $\mathbf{M}_{s,p} \leftarrow \texttt{MEncode}(ct_{\text{cmp}_{s,p}}, \mathbf{RK}, N, s, p)$
    $ct_{\text{res}} \leftarrow \texttt{HomOrdApp}(ct_{\text{res}}, \mathbf{M}_{s,p}, |\mathcal{T}|_{\text{row}}, s, p, \mathbf{RK})$
Return : ct_{res}

```sql
SELECT * FROM
    (SELECT *, DISTANCE(T_1.feature,
     vq1_feature) AS dist FROM T_1)
AS subquery
ORDER BY dist ASC
LIMIT 4;
```

(a) VQ1.

```sql
SELECT id, name, color, price FROM
    (SELECT *, DISTANCE(T_1.feature,
     vq2_feature) AS dist FROM T_1)
AS subquery
ORDER BY dist DESC
LIMIT 16;
```

(b) VQ2.

```sql
SELECT id,
    AVG(price) AS avg_price, COUNT(*) AS id_count,
    SUM(sales) AS sum_sales, SUM(quantity) AS sum_ quant,
    DISTANCE(Table_3.feature, hq1_feature) AS dist
FROM T_3
WHERE T_3.price < 100
    AND (T_3.color = 'red' OR T_3.color = 'blue')
    AND T_3.shipdate >= '2024-01-01'
GROUP BY id
ORDER BY dist;
```

(c) HQ1.

```sql
SELECT id,
    SUM(price*(1-discount)) AS revenue,
    COUNT(*) AS id_count, SUM(sales) AS sum_sales,
    AVG(discount) AS avg_discount,
    DISTANCE(Table_3.feature, hq2_feature) AS dist
FROM T_3
WHERE T_3.discount > 0.25 AND T_3.quantity >= 2
    AND (T_3.sales > 100 AND T_3.sales < 500)
GROUP BY id
ORDER BY dist
LIMIT 4;
```

(d) HQ2.

Appendix Figure C1: The detail vectorized query and hybrid query benchmarks in Section 6.3.

in their test to complete the synchronization. For the Top-k benchmark, we set the input length to 256 and the values of k to 1, 4, 8, and 16, respectively.

C.2 SQL Benchmark Details

For TPC-H benchmark [34], we remove the `JOIN` conditions to be consistent with [10] and [95]. We give the vectorized query benchmarks VQ1 and VQ2 in Figure C1a and Figure C1b, as well as the hybrid query benchmarks HQ1 and HQ2 in Figure C1c and Figure C1d. For hybrid query tests, since [10] and [95] do not support vector distance calculation, we remove the `Distance` operator for fair comparison.

Algorithm 11: Homomorphic Top-k `HomTopK`

Input : An unsorted QRLWE ciphertext $ct_{0,0}$, rotate key **RK**, precision parameter (Θ, α), and k.
Output : A QRLWE ciphertext ct_{res}, where the first k elements are the Top-k results.

$ct_{\text{res}} \leftarrow ct_{0,0}$
for $s = 0$ **to** $\log_2 k$ **do**
  for $p = 0$ **to** s **do**
    $ct_{\text{cmp}_{s,p}} \leftarrow \texttt{HomOrdGen}(ct_{\text{res}}, N, s, p, \mathbf{RK}, (\Theta, \alpha))$
    $\mathbf{M}_{s,p} \leftarrow \texttt{MEncode}(ct_{\text{cmp}_{s,p}}, N, s, p, \mathbf{RK})$
    $ct_{\text{res}} \leftarrow \texttt{HomOrdApp}(ct_{\text{res}}, \mathbf{M}_{s,p}, N, s, p, \mathbf{RK})$
for $i = 0$ **to** $\log_2(N) - \log_2(k) - 1$ **do**
  $s \leftarrow \log_2(k)$
  $Step \leftarrow 2^s + (2^{i+1} - 1) \cdot 2k$
  for $i = 0$ **to** N **step** 2^{s+2} **do**
    $\mathbf{Mask}[i : i + 2^{s+1}] \leftarrow 1$
    $\mathbf{Mask}[i + 2^{s+1} : i + 2^{s+2} - 1] \leftarrow -1$
  for $i = 0$ **to** N **step** 2^{s+1} **do**
    $\mathbf{Eli}[i : i + 2^s] \leftarrow 1$
    $\mathbf{Eli}[i + 2^s : i + 2^{s+1} - 1] \leftarrow 0$
  for $i = 0$ **to** $w - 1$ **do**
    $ct_{\text{mask}}[i] \leftarrow ct_{\tilde{x}}[i] \cdot \mathbf{Mask}$
    $ct_{\text{rot}}[i] \leftarrow \texttt{Rotate}(ct_{\text{mask}}[i], \mathbf{RK}, Step)$
  $ct_{\text{cmp}_{s,0}} \leftarrow \texttt{ArbQuantComp}(ct_{\text{mask}}, ct_{\text{rot}}, >, (\Theta, \alpha)) \cdot \mathbf{Eli}$
  $ct_{\text{cmp}_{\text{diag}}} \leftarrow ct_{\text{cmp}_{s,0}} + \texttt{Rotate}(ct_{\text{cmp}_{s,0}}, \mathbf{RK}, -Step)$
  $ct_{\text{cmp}_{\text{sub}}} \leftarrow \texttt{HomGate}(ct_{\text{cmp}_{s,0}}, \texttt{NOT})$
  $ct_{\text{cmp}_{\text{sup}}} \leftarrow \texttt{Rotate}(ct_{\text{cmp}_{\text{sub}}}, \mathbf{RK}, N - Step)$
  for $i = 0$ **to** $w - 1$ **do**
    $ct_{\text{sup}}[i] \leftarrow \texttt{Rotate}(ct_{\text{res}}[i], \mathbf{RK}, N - Step)$
    $ct_{\text{sub}}[i] \leftarrow \texttt{Rotate}(ct_{\text{res}}[i], \mathbf{RK}, Step)$
    $ct_{\text{res}}[i] \leftarrow ct_{\text{sup}}[i] \cdot \mathbf{M}_{s,0}[0] + ct_{\text{res}}[i] \cdot \mathbf{M}_{s,0}[1] + ct_{\text{sub}}[i] \cdot \mathbf{M}_{s,0}[2]$
  for $p = 1$ **to** $\log_2 k$ **do**
    $ct_{\text{cmp}_{\log_2 k,p}} \leftarrow \texttt{HomOrdGen}(ct_{\text{res}}, N, \log_2 k, p, \mathbf{RK}, (\Theta, \alpha))$
    $\mathbf{M}_{\log_2 k,p} \leftarrow \texttt{MEncode}(ct_{\text{cmp}_{s,p}}, N, s, p, \mathbf{RK})$
    $ct_{\text{res}} \leftarrow \texttt{HomOrdApp}(ct_{\text{res}}, \mathbf{M}_{\log_2 k,p}, N, \log_2 k, p, \mathbf{RK})$
Return : ct_{res}