

SmartPIR: A Private Information Retrieval System using Computational Storage Devices

Zehao Chen
School of Cyber Science and
Technology
Shandong University
Qingdao, China
Quan Cheng Laboratory
Jinan, China
202221102@mail.sdu.edu.cn

Honghui You
School of Cyber Science and
Technology
Shandong University
Qingdao, China
State Key Laboratory of
Cryptography and Digital Economy
Security
Shandong University
Qingdao, China
202117073@mail.sdu.edu.cn

Qian Wei
School of Computer Science and
Technology
Shandong University
Qingdao, China
202220771@mail.sdu.edu.cn

Hang Lu
SKLP, Institute of Computing
Technology, Chinese Academy of
Sciences
Beijing, China
Zhongguancun Laboratory
Beijing, China
luhang@ict.ac.cn

Lei Ju*
Quan Chen Laboratory
Jinan, China
School of Cyber Science and
Technology
Shandong University
Qingdao, China
julei@sdu.edu.cn

Zhaoyan Shen
School of Computer Science and
Technology
Shandong University
Qingdao, China
shenzhaoyan@sdu.edu.cn

Abstract

Fully Homomorphic Encryption-based Private Information Retrieval systems provide strong privacy by enabling encrypted queries on databases hosted by untrusted servers. However, adoption is limited by system-level bottlenecks, including severe I/O constraints in large-scale settings and inefficiencies in handling variable-length data. To solve these issues, we present SmartPIR, a scalable PIR system that tackles these challenges through a protocol and architecture co-design. First, we introduce an in-storage computing framework that offloads FHE operations to computational storage devices (CSDs), eliminating the overhead of large-volume data movement. To address the limited computational capacity of CSDs, we further propose a zero-skipping encoding strategy at the protocol level, which decouples actual payloads from paddings to avoid redundant computations. Additionally, SmartPIR incorporates two key architectural optimizations: (1) a resource-efficient FPGA circuit design and (2) a load-aware scheduling strategy, which collectively sustain high throughput and ensure scalability. Implemented on a commercial off-the-shelf CSD array, SmartPIR achieves a $10^2 \times \sim 10^3 \times$ speedup over state-of-the-art CPU-based PIR schemes.

*Corresponding author

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
MICRO '25, Seoul, Republic of Korea

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-1573-0/25/10
<https://doi.org/10.1145/3725843.3756060>

CCS Concepts

• **Hardware** → **Communication hardware, interfaces and storage**; • **Security and privacy** → **Security services**.

Keywords

Private Information Retrieval, Fully Homomorphic Encryption, Computational Storage Devices

ACM Reference Format:

Zehao Chen, Honghui You, Qian Wei, Hang Lu, Lei Ju, and Zhaoyan Shen. 2025. SmartPIR: A Private Information Retrieval System using Computational Storage Devices. In *58th IEEE/ACM International Symposium on Microarchitecture (MICRO '25)*, October 18–22, 2025, Seoul, Republic of Korea. ACM, New York, NY, USA, 14 pages. <https://doi.org/10.1145/3725843.3756060>

1 Introduction

Private Information Retrieval (PIR) [14] is a privacy-preserving technology that allows clients to retrieve specific data entries from a public database without disclosing their interest. With the widespread adoption of cloud computing and the growing dominance of the data explosion [15, 32, 33], PIR has become increasingly essential in privacy-sensitive domains such as finance and healthcare [18, 62], where minimizing data exposure is crucial.

PIR technology employs an all-for-one model. In a PIR system, each query triggers a cryptographic processing step where all database entries must be computed. This approach ensures that query access patterns remain concealed. Modern PIR systems increasingly integrate Fully Homomorphic Encryption (FHE) [4, 25, 57] as the computational backbone, enabling the server to process encrypted queries on the public database and return encrypted results without ever decrypting or learning the query contents. Unlike earlier

information-theoretic PIR protocols [46], which rely on multiple non-colluding servers, FHE-based PIR can operate securely in a single-server setting, significantly broadening its application.

Although FHE-based PIR systems provide strong privacy guarantees, they suffer from significant performance bottlenecks [4, 37]. These limitations arise primarily from the high computational overhead of FHE operations, which ultimately degrade end-to-end throughput and increase latency. To solve this, prior research optimized PIR designs at both the protocol and system levels. At the protocol level, efforts have focused on algorithmic improvements such as advanced encoding techniques [6, 7], query batching [40], and recursive retrieval methods [34, 39] to reduce algorithmic costs. At the system level, performance was improved by offloading compute-intensive FHE operations to specialized hardware, such as GPUs [21, 43], thereby enhancing the overall efficiency.

Despite the considerable progress made by prior works, we observe that the performance and scalability of PIR systems remain fundamentally constrained by system-level inefficiencies arising from tremendous data movement and high computational demand. First, despite hardware acceleration for FHE operations has achieved significant speedups, these improvements have not translated proportionally to overall system throughput. Our analysis reveals an emerging architectural bottleneck: as computation becomes faster through hardware acceleration, I/O overhead now dominates end-to-end latency. For example, our experiments on the A100 GPU reveal that I/O operations account for more than 85% of total query latency (see § 3). Second, most existing PIR protocols [4, 6, 7, 39] typically employ padding strategies to ensure uniform entry sizes across the database (e.g., by prepending zeros to shorter entries to equalize their lengths with the longest one). While this design simplifies the protocol design, it incurs substantial computational overhead by requiring the system to process redundant padding data alongside valid payloads. This inefficiency is particularly pronounced in real-world datasets characterized by high variability in entry lengths [8, 10], where padding can inflate query latency by up to 1~2 orders of magnitude (see § 3).

In this work, we leverage emerging Computational Storage Devices (CSDs) to simultaneously address the dual challenges of excessive data movement and computational overhead. Embedding programmable compute resources (e.g., FPGAs) directly within storage devices—as in platforms like Samsung’s SmartSSD [30]—enables in-situ data processing, significantly reducing data movement and accelerating computation. Previous research has demonstrated the effectiveness of CSDs in accelerating various data-intensive workloads, including machine learning applications [27, 55, 56] and key-value store operations [20].

However, deploying PIR systems on CSDs is non-trivial since the computation capability inside a CSD is limited. Unlike the compute-light workloads in prior works (e.g., filter [20, 27], aggregator [28]), FHE operations in PIR systems are relatively compute-intensive. Hence, it is challenging to deploy the full PIR pipeline on resource-constrained FPGAs while maintaining computational efficiency. Moreover, the computation expansion brought by variable-length data further complicates the design. As previously noted, conventional PIR protocols operate on uniform-length data layouts, necessitating the padding of shorter entries. Although relocating computation to CSDs can mitigate I/O-related delays, it does not eliminate

the overhead introduced by padding. As a result, the throughput of PIR workloads with high variability in entry lengths remains constrained. Meanwhile, achieving scalability in CSD-based architectures requires careful system design. Effectively supporting PIR across multiple CSDs requires coordinated data partitioning, balanced scheduling of homomorphic operations, and minimal inter-device communication. Naive multi-device deployment can lead to performance imbalance and limited scalability.

This paper presents SmartPIR, a scalable FHE-based PIR system that addresses the aforementioned challenges through three key innovations. First, we introduce an in-storage computing framework that offloads FHE operations to computational storage devices (CSDs), effectively eliminating overhead associated with large-scale data movement. Second, to overcome the computational constraints of CSDs, we develop the SmartPIR protocol featuring a zero-skipping encoding scheme that separates valid payloads from paddings, thereby avoiding redundant computations on null entries during encrypted query processing. This approach significantly reduces the computational overhead induced by variable-length records. We also propose a novel folding retrieval mechanism that optimizes the computation-communication trade-off, maintaining practical query sizes (within several megabytes) while introducing only marginal computational overhead. Finally, we design a protocol-optimized FHE hardware circuit design that maximizes FPGA resource utilization, and a distributed load-aware scheduler that ensures efficient task distribution across multiple CSDs. Together, these innovations enable SmartPIR to achieve both system-level scalability and high performance.

In summary, the key contributions of this paper are as follows:

- We present SmartPIR, an efficient and scalable PIR system that leverages commercial CSDs to offload both computation and I/O intensive FHE operations, fundamentally addressing the I/O bottleneck in conventional PIR implementations.
- We develop a novel SmartPIR protocol featuring: (1) zero-skipping encoding that eliminates redundant computation on padding data, and (2) a folded retrieval mechanism that maintains practical communication overhead while efficiently supporting variable-length database entries.
- We introduce a protocol-aware FHE hardware circuit design that maximizes FPGA resource efficiency, coupled with an intelligent load-aware scheduling strategy, to jointly achieve high throughput and system scalability.
- We develop the fully functional prototype based on the Microsoft SEAL library [51] and SmartSSDs [30]. Our experimental results on real-world workloads show that SmartPIR achieves a $10^2 \times \sim 10^3 \times$ speedup over the state-of-the-art PIR scheme deployed on CPU platforms.

2 Background

2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE), based on the Ring Learning With Errors (RLWE) problem [48], allows computation on encrypted data without decryption, positioning it as a fundamental primitive for Private Information Retrieval (PIR) systems. Among the various FHE schemes, the BFV scheme [25] is commonly utilized in PIR research. This work also focuses on the BFV scheme.

In the BFV scheme, a plaintext pt is represented as a vector of length N , where N is a power of two ($2^{10}, 2^{11}, \dots, 2^{15}$), and each element is an integer modulo a prime T . The plaintext is the basic unit for BFV operations. Using a public key pk , the plaintext pt is encrypted into a ciphertext $ct = \text{Enc}(pt) = (a, b)$, where a and b are vectors of length N with coefficients modulo Q , with Q larger than T for security. The plaintext can be recovered from ct using the private key sk . Given a $pt = [1, 2, 3, 4]$ (assume $N = 4$) and its corresponding encryption $ct = \text{Enc}(pt)$, the following BFV operations are supported:

- **Add**: Adds two ciphertexts by performing element-wise addition over their underlying plaintext vectors. For example, $ct + ct = (a + a, b + b)$, which represents the encryption of $[2, 4, 6, 8]$.
- **PMult**: Multiplies a ciphertext by a plaintext vector via element-wise multiplication. The dimensionality of the encrypted result remains unchanged. For example, $ct \odot pt = (a \odot pt, b \odot pt)$, which represents the encryption of $[1, 3, 6, 9]$.
- **Mult**: Multiplies two ciphertexts via element-wise multiplication, and the result is a ciphertext with three dimensions. Specifically, the operation $ct \odot ct$ results in $(a \odot a, b \odot b, 2a \odot b)$, with a cross-product term. This increase in dimensionality introduces computational overhead and the need for subsequent reduction. To restore the ciphertext to its original two-dimensional form and improve efficiency, a KeySwitch operation is applied. This operation transforms the ciphertext back into two dimensions, preserving the underlying plaintext meaning while reducing complexity.
- **Rotate**: Rotates a ciphertext by k positions within the plaintext vector. This is implemented via an automorphism operation followed by KeySwitch, enabling encrypted cyclic shifts without decryption. For example, rotating ct by 1 position yields $\phi_1(ct)$, which represents the encryption of $[2, 3, 4, 1]$.

2.2 Private Information Retrieval System

A PIR system comprises three primary components: the client, the server, and the server-side database $D = \{d_1, d_2, \dots, d_n\}$. The PIR server allows a client to retrieve the i -th entry from the database without revealing the index i . A simple FHE-based PIR scheme usually consists of four steps:

- **Setup**: The client and server agree on BFV parameters (e.g., N, T, Q), index information, and database size n .
- **QueryGen**: To query the k -th entry, the client generates n encrypted queries $\{\delta_1, \delta_2, \dots, \delta_n\}$ and sends them to the server, where $\delta_k = \text{Enc}([1, \dots, 1])$ and $\delta_i = \text{Enc}([0, \dots, 0])$ for $i \neq k$.
- **AnswerGen**: Upon receiving the encrypted query, the server processes the database homomorphically. First, it encodes all entries $\{d_i\}$ in the database into plaintext vectors $\{pt_i\}$ to conform to the input requirements of the BFV scheme. To ensure correct indexing, especially when handling variable-length data, the server pads all entries to match the longest one [14, 37]. Figure 1 illustrates the encoding result for 8 variable-length entries, where $d_1 \sim d_4$ are padded to 64 bits to ensure alignment between entries and queries. Second, the server computes a homomorphic inner product between queries and entries:

$$Ans = \sum_{i=1}^n \delta_i \odot pt_i = \text{Enc}(d_k)$$

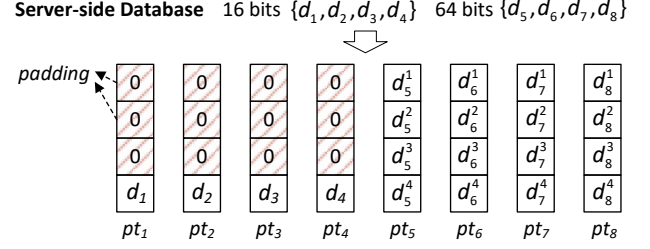


Figure 1: Padding overview, where $N = 4$ and $T = 2^{16} + 1$, and each d_i^j corresponds to a 16-bit segment.

- **Recover**: Finally, the client receives the Ans and decrypts it using its private key sk to recover d_k .

The PIR server must scan the entire database, resulting in substantial computational, communication, and I/O costs. These costs grow linearly with the database size, i.e., $O(n)$, posing significant performance challenges. To this end, a variety of optimization strategies have been proposed. At the protocol level, SealPIR [7] introduces a lightweight query expansion technique that expands a one-hot ciphertext to N queries, reducing communication overhead. BatchPIR [40] improves performance in high-throughput retrieval scenarios by amortizing expensive BFV operations across multiple queries. MulPIR [6] further advances prior work by leveraging multiplicative homomorphism to enable a tunable trade-off between communication and computation, achieving state-of-the-art efficiency in large-scale scenarios. At the system level, works such as PIRonGPU [43] offload the compute-intensive BFV operations on GPU platforms, helping to bridge the performance gap between encrypted and plaintext computations. By integrating such GPU-accelerated primitives, PIR systems achieve improvements in both latency and throughput.

Despite recent advancements, two key system-level challenges remain unresolved. First, existing FHE-based PIR schemes exhibit inefficiencies when processing variable-length data entries. To maintain alignment, all entries must be padded to a uniform size, leading to inflated data volumes and increased computational overhead. Second, full-database scans impose substantial I/O overhead, with performance heavily constrained by bandwidth limitations and access latency of storage devices (e.g., SSDs). These challenges motivate our design and are further discussed in detail in § 3.

2.3 Computational Storage Device

Computational storage devices (CSDs) have been utilized to alleviate I/O bottlenecks in modern computing systems [34, 58, 59, 64]. By embedding computational engines directly within storage devices, CSDs significantly mitigate data movement overheads, thereby enhancing overall system efficiency. In this paper, we adopt the CSD design of a commercial product of Samsung, SmartSSD [30].

Compared to conventional NVMe SSDs, SmartSSD features two key enhancements. First, it incorporates an internal PCIe switch to enable P2P communication between the NVMe SSDs and the onboard accelerator. This design eliminates data transfers through the host's PCIe bus and mitigates the I/O bottlenecks typically encountered in traditional storage architectures. Second, SmartSSD integrates a lightweight FPGA accelerator that processes data directly

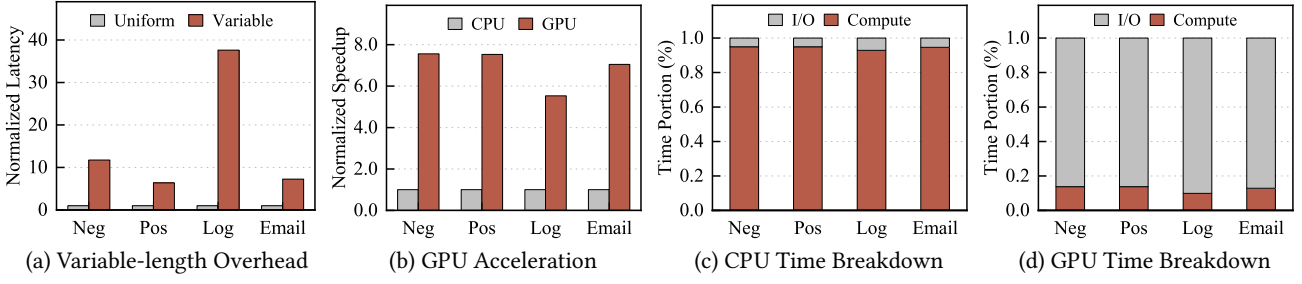


Figure 2: (a) Retrieval time increases from the uniform-length to variable-length datasets. (b) Speedup from the GPU acceleration. (c) CPU time breakdown. (d) GPU time breakdown. The experimental setting follows § 8.

at the storage layer. By carrying out computations closer to the data, this method reduces the host’s workload and increases overall throughput through more efficient data movement. Moreover, the computational capacity and bandwidth scale linearly with the number of SmartSSDs deployed. Multiple studies [20, 27, 28, 55, 56] have demonstrated this scalability, underscoring the effectiveness of SmartSSD in handling I/O-intensive workloads.

3 Motivation

3.1 Inefficient Variable-length Handling

As noted in § 2.2, conventional FHE-based PIR schemes typically enforce database structural alignment by padding each entry to match the length of the longest one. This approach introduces substantial computational redundancy, particularly for datasets with high entry-length variance. To quantify this inefficiency, we evaluate the CPU-based MulPIR protocol [6] across four real-world datasets—*Pos*, *Neg*, *Log*, and *Email* (details in Table 2)—where entry lengths vary significantly due to semantic structure (e.g., timestamped logs, natural language text, etc.). As a comparison, we construct corresponding uniform-length versions of these datasets by padding or truncating the entries to their mean bitwidths, while keeping the total number of entries and total data volume fixed.

Figure 2(a) illustrates the query latency of processing variable-length data versus uniform-length datasets. *Uniform* refers to the padded, uniform-length versions of the four datasets, while *Variable* represents their original variable-length form. All results are normalized to the uniform-length baseline. The analysis reveals that padding incurs substantial performance penalties, increasing latency by 12.74×, 6.37×, 38.62×, and 8.26× for *Pos*, *Neg*, *Log*, and *Email*. The latency increase correlates with the bitwidth variance among entries within the datasets, as higher variance amplifies padding overhead and leads to greater computational workload.

These results underscore the susceptibility of existing PIR protocols to structural irregularity, revealing that padding-induced inefficiencies represent a critical performance bottleneck. *These findings demonstrate the necessity for novel PIR protocols capable of efficient variable-length data processing without incurring the substantial computational overhead associated with padding* (§ 5).

3.2 Dominant I/O Overhead

Hardware acceleration, particularly through GPUs, has significantly advanced FHE computation. For example, the HEonGPU

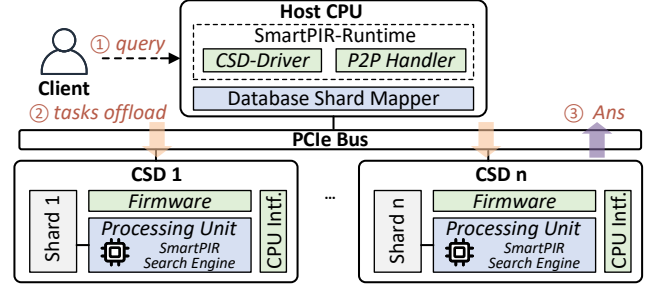


Figure 3: The SmartPIR framework overview.

library [43] implements the BFV scheme on NVIDIA A100 GPUs, achieving 2~3 orders of magnitude speedup for individual BFV operations. Building on these results, we adapted the MulPIR to the A100 GPU and evaluated its performance using uniform-length versions of the datasets to isolate the effects of padding overhead.

Figure 2(b) presents the normalized speedup of our GPU implementation relative to the CPU baseline for single-query execution. While computational latency improved by 7.6×, 7.5×, 5.5×, and 7.1× for the *Pos*, *Neg*, *Log*, and *Email* datasets, respectively, the overall end-to-end speedup was markedly lower—falling well short of the performance gains for different BFV operations.

To identify the performance bottleneck, we conducted execution time profiling across CPU and GPU implementations (Figures 2(c) and 2(d)). Our analysis reveals a substantial increase in I/O overhead—from 7% of total runtime in the CPU baseline to over 85% in the GPU-accelerated implementation. This shift is driven by two key factors: (1) Hardware Acceleration Unmasks I/O Bottleneck: The GPU’s high compute throughput shortens processing time, exposing I/O as the new critical path and shifting the system bottleneck from computation to data movement; (2) Extended Data Transfer Path: The GPU implementation introduces additional data transfer steps (SSD → CPU → GPU), replacing the CPU’s direct SSD access and significantly increasing end-to-end latency.

These findings reveal fundamental scalability limitations in existing PIR architectures imposed by I/O bottlenecks. *This analysis motivates our design of a co-optimized PIR architecture that holistically integrates computation and data movement, ensuring computational efficiency while overcoming I/O constraints* (§ 6).

4 SmartPIR Framework Overview

SmartPIR is a full-stack PIR system that integrates customized protocol and hardware designs within an in-storage computing framework, enabling scalable and privacy-preserving retrieval over practical variable-length datasets. The system is built atop an array of computational storage devices (e.g., SmartSSD) with embedded FPGAs capable of executing FHE operations directly within the storage layer. As shown in Figure 3, these CSDs are connected to a host CPU over PCIe, forming a host-device architecture that partitions responsibility across layers. The host handles orchestration and client communication, while each CSD stores a database shard and executes PIR queries locally through its embedded FPGA engine.

System Workflow. With SmartPIR, the execution of a PIR query proceeds in three coordinated stages: ① **Receiving Query:** The host CPU receives encrypted queries from the clients and dispatches them to the CSDs. ② **Tasks Offload:** Upon receiving queries, each CSD independently executes the PIR query over its local encrypted shard, accessing flash data directly via P2P channels without host-side intervention. ③ **Generating Answer:** After computation, each CSD returns a partial encrypted answer to the host, which aggregates the results to generate the final response.

The above architecture offers two key benefits: (1) By pushing most PIR computational workloads to the storage layer, it minimizes data transfer between SSDs and the host, thus improving I/O efficiency and alleviating pressure of the host CPU; (2) As each CSD operates independently with its own FPGA and bandwidth path, it achieves linear scalability to the number of CSDs. However, due to the inherent computation constraints of CSDs, it is challenging to achieve SmartPIR's performance and scalability goals. First, variable-length datasets introduce structural irregularity that prior PIR protocols cannot efficiently handle, leading to excessive padding and wasted computation. Second, while CSDs provide near-data processing, their limited FPGA resources make it challenging to support a full-fledged and efficient PIR engine without careful hardware specialization. Third, entries-length variability leads to unbalanced workloads across CSDs. Without considering this heterogeneity, simple partitioning strategies (e.g., striping or round-robin [66]) can cause device underutilization and degrade parallel efficiency. To address these limitations, we further integrate three key techniques:

- **SmartPIR Protocol.** We introduce the SmartPIR protocol that eliminates padding overhead using zero-skipping encoding and folding retrieval strategies (§ 5).

- **Resource-efficient Circuit.** We design a modular hardware engine optimized for FPGA-constrained environments, enabling full in-storage PIR execution (§ 6).

- **Load-balancing.** We implement a load-aware scheduler that balances PIR workloads under skew entry lengths (§ 6).

5 SmartPIR Protocol Design

In this section, we introduce a zero-skipping encoding scheme to support variable-length data handling (§ 5.1), propose protocol-level optimizations to reduce communication costs (§ 5.2), and present the complete SmartPIR workflow with its core components and interactions (§ 5.3).

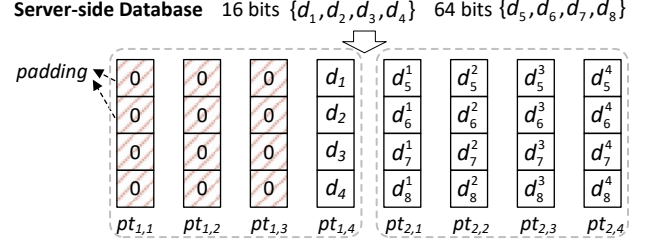


Figure 4: ZSE procedure. $N = 4$ and $T = 2^{16} + 1$. Plaintext vectors such as $pt_{1,1} \sim pt_{1,3}$ are zero vectors.

5.1 Zero-Skipping Encoding

We present **Zero-Skipping Encoding (ZSE)**, a novel encoding scheme that eliminates computational waste in variable-length PIR processing by structurally separating meaningful payload from padding. This decoupling enables servers to selectively process only operationally relevant data while preserving protocol correctness. Suppose the server database contains βN entries, where each entry has variable length with a maximum of l bits. ZSE transforms these entries through two steps:

- **Padding and Partition:** Similar to traditional approaches, ZSE initially pads each of the βN entries to a standardized bitwidth l bits. The server then performs a bitwidth-aware sorting of the entries based on their original lengths, followed by partitioning into β uniform groups of size N . This optimized grouping strategy collocates entries with similar padding ratios, enabling efficient batch processing of padding-dominated entries in subsequent operations.

- **Horizontal Encoding:** Following partitioning, the server applies a horizontal encoding scheme to transform entries into plaintext vectors. Unlike conventional approaches that encode multiple entries into a single plaintext, our method strategically distributes each entry across multiple vectors. Given the BFV plaintext modulus T , the N entries in a partition are decomposed into α chunks, where $\alpha = \lceil l / \log_2(T - 1) \rceil$ and each chunk is encoded into a plaintext vector. This horizontal organization maintains the one-to-one mapping between original entries and their encoded positions while structurally segregating padding bits (high-order zeros) from payload data. Consequently, for partitions containing entries with a bitwidth smaller than l bits, the paddings will be aggregated into vectors that do not contain any payload, denoted by zero vectors. This provides an opportunity for the server to identify and bypass those all-zero vectors algorithmically, eliminating unnecessary computations while preserving operational correctness.

Figure 4 demonstrates the ZSE scheme applied to eight variable-length entries. The encoding process yields: Partition 1 (entries $d_1 \sim d_4$). Encoded as plaintext vectors $pt_{1,1} \sim pt_{1,4}$, where $pt_{1,1} \sim pt_{1,3}$ are zero vectors (containing only padding); Partition 2 (entries $d_5 \sim d_8$): Represented as $pt_{2,1} \sim pt_{2,4}$. When serving a PIR query, it requires the client to send an encrypted query per partition. For example, to retrieve the 6-th entry, the clients sends two ciphertexts: $\delta_1 = \text{Enc}([0, 0, 0, 0])$ (for Partition 1) and $\delta_2 = \text{Enc}([0, 1, 0, 0])$ (for Partition 2). The encrypted answer Ans is then computed as:

$$Ans[j] = \sum_{i=1}^{\beta} pt_{i,j} \odot \delta_i, j \in [0, \alpha - 1]. \quad (1)$$

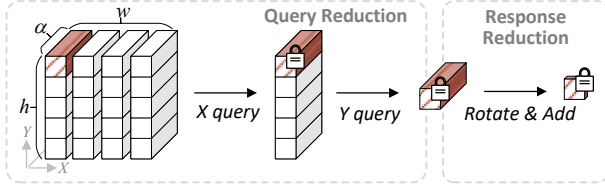


Figure 5: The overview of folding retrieval.

In this process, computations involving zero vectors (e.g., $pt_{1,1} \sim pt_{1,3}$) are skipped. Crucially, this skipping mechanism does not expose any information about the client's query. This optimization preserves query privacy as vector skipping depends solely on the server's static data organization, remaining independent of query parameters and leaving no observable computation traces. A detailed security analysis is provided in § 5.4.

5.2 Folding Retrieval

Although the ZSE method enables the PIR server to handle variable-length entries more efficiently, it also incurs certain trade-offs. Specifically, the ZSE method splits and distributes each entry across several plaintexts, increasing the complexity of the server's retrieval process and leading to higher communication overhead. Consider a database D with $n = \beta N$ variable-length entries, where the maximum length of each entry is $l = \alpha \times \log_2(T - 1)$ bits. According to the formula 1, the ZSE method results in both $O(n/N)$ client queries and $O(\alpha)$ server responses.

To mitigate these issues, we leverage the idea of folding [6, 7] to reduce the query size by narrowing the search space. Specifically, after ZSE encoding, the database is reorganized into a three-dimensional matrix, where β plaintext groups (each containing α plaintexts) are arranged in the X-Y plane, with α plaintexts stacked along the Z axis. As shown in Figure 5, this transformation reduces the search space from a 1-D index of size $1 \times n/N$ to a 2-D index of size $\sqrt{n/N} \times \sqrt{n/N}$. In this layout, the client first sends **X queries** (in the form of $\text{Enc}([0, \dots, 0])$ or $\text{Enc}([1, \dots, 1])$) to locate the specific column. Then, **Y queries** are sent to identify the target ciphertext group and retrieve the target entry. Since the search space is reduced from $1 \times n/N$ to $\sqrt{n/N} \times \sqrt{n/N}$, this reduces query complexity from $O(n/N)$ to $O(\sqrt{n/N})$. Importantly, the form of X queries remains consistent with traditional queries, making it compatible with the Expand algorithm from SealPIR [7], which can expand a ciphertext $\text{Enc}(x^l)$ to N ciphertexts (with the i -th as $\text{Enc}([1, \dots, 1])$ and the rest as $\text{Enc}([0, \dots, 0])$). Therefore, by adjusting the column-to-row ratio in the X-Y plane, the query size can be further optimized. When the $w : h$ ratio is $N : 1$, the query complexity is reduced to $O(\sqrt{n/N})$.

To further reduce the response size, we aggregate all encrypted ciphertexts into a single ciphertext. The key observation is that the encrypted responses are one-hot vectors; that is, for each $\text{Ans}[j]$, $j \in [0, \alpha]$ in Equation 1, each vector of degree N only contains one non-zero element, with all other elements being zero. This sparsity allows for more efficient space utilization. By applying the Rotate and Add operations, we can shift and accumulate all non-zero payloads into a single ciphertext using the following

Algorithm 1: SmartPIR :: QueryGen

Input: Target idx

```

1  $oft \leftarrow idx \% N$  // Compute offset
2  $h \leftarrow \sqrt{\beta/N}, w \leftarrow h \cdot N$ 
3  $r\_idx \leftarrow idx / N / w$  // row index
4  $c\_idx \leftarrow idx / N \% w$  // column index
5  $X[c\_idx / N] \leftarrow \text{Enc}(x^{c\_idx \% N})$ 
6  $Y[r\_idx] \leftarrow \text{Enc}(x^{oft})$  // Other are  $\text{Enc}([0, \dots])$ 
Output:  $X[w/N]$  &  $Y[h]$ 

```

Algorithm 2: SmartPIR :: AnswerGen

Input: $D, h, w, \alpha \leftarrow l/\log_2(T - 1), X[w/N], Y[h]$

```

1  $X'[w] \leftarrow \text{SealPIR}::\text{Expand}(X[w/N])$ 
2  $M[h][w][\alpha] \leftarrow \text{ZSE}(D)$ 
3 for  $k \leftarrow 0$  to  $\alpha - 1$  do
4   for  $j \leftarrow 0$  to  $h - 1$  do
5     for  $i \leftarrow 0$  to  $w - 1$  do
6       if  $M[j][i][k]$  is not zero vector then
7          $T[j][k] \leftarrow T[j][k] + M[j][i][k] \odot X'[i]$ 
8 for  $k \leftarrow 0$  to  $\alpha - 1$  do
9   for  $j \leftarrow 0$  to  $h - 1$  do
10     $T'[k] \leftarrow T'[k] + T[j][k] \odot Y[j]$ 
11 for  $k \leftarrow 0$  to  $\alpha - 1$  do
12    $\text{Ans} \leftarrow \text{Ans} + \text{Rotate}(T'[k], k)$ 
Output: Ans

```

technique:

$$\text{Ans} = \sum_{a=1}^{\alpha} \text{Rotate}(\text{Ans}[a], a - 1)$$

In this way, the response from server to client is reduced from $O(\alpha)$ to $O(\lceil \alpha/N \rceil)$.

5.3 SmartPIR Protocol Workflow

In this section, we describe the detailed procedure of the SmartPIR protocol. Algorithmically, SmartPIR builds on the classical homomorphic inner-product-based PIR framework, while introducing two key innovations: (1) Unlike MulPIR, which performs homomorphic multiplications on padded zero entries, ZSE eliminates most of the redundant operations introduced by paddings, thereby improving the system efficiency. (2) Instead of relying on traditional inner products for result aggregation, SmartPIR adopts a lightweight Rotate-Add fusion strategy, enabling the server to return a ciphertext containing only the target record. Suppose a server-side database D contains βN variable-length entries, each of up to l bits. The SmartPIR protocol workflow involves four routines as follows:

- **Setup:** The server and client initialize shared parameters, including BfV parameter N, T, Q , database size n , the maximum bandwidth l , and the index.

- **QueryGen:** As described in Algorithm 1, the client encodes the target index idx into two sets of encrypted one-hot vectors, X and Y , which correspond to column and row selection within

the server's folded 3-D matrix. Specifically, X identifies the target column block, and Y pinpoints the row and offset within that block. All non-target slots in X, Y are filled with encryptions of zeros, preserving query indistinguishability.

- **AnswerGen:** Upon receiving the encrypted query X, Y , the server performs the PIR computation in four steps: (1) **Expand**: it expands $X[w/N]$ to $X'[w]$ via the Expand algorithm (line 1 in Algorithm 2). (2) **PMAC** (PMult Accumulation) : it uses the ZSE method to encode all entries into plaintext vector in M (line 2), then computes the inner product of M with X' to locate the target column, skipping all zero vectors during this step (line 3~7). (3) **MAC** (Mult Accumulation): it further multiplies T with Y (line 8~10) to locate the target partition. (4) **RAC** (Rotate Accumulation): it aggregates the result into a single ciphertext Ans via Rotate and Add (line 11~12), and sends it to the client. Notably, in the SmartPIR framework described in § 4, Step 1 is executed on the host due to its negligible computational overhead. In contrast, Steps 2~4, which directly operate on the dataset, are offloaded to the CSDs.

- **Recover:** The client decrypts Ans to recover d_{idx} .

5.4 Security Analysis

We analyze the privacy guarantees of SmartPIR under a semi-honest adversarial model, where the server follows the protocol but may attempt to infer the client's query index from observable computations. Our argument relies on the semantic security of the BFV scheme, which is IND-CPA secure [13]. SmartPIR achieves query privacy through the following properties:

- **Data Confidentiality.** All operations are performed over ciphertexts, ensuring that the server learns nothing about the query or accessed entry due to the semantic security of the BFV scheme.
- **Indistinguishable Computation.** While SmartPIR optimizes processing by skipping computations on zero vectors for variable-length entries, this skipping pattern is dictated solely by the server-side database layout, not the client's query. This design ensures that every query triggers an identical access and computation flow from the server's perspective, effectively preventing any query-dependent branching or data-dependent timing behavior.
- **Indistinguishable Query & Response.** The client always generates a fixed number of ciphertexts per query— $X[w/N]$ and $Y[h]$ —regardless of the bitwidth of the requested entry. Similarly, the server returns a response of fixed size, $Ans[\alpha/N]$, independent of the query. This uniform pattern ensures that the query/response shape is indistinguishable.

6 SmartPIR Hardware Design

In this section, we detail the design of the SmartPIR hardware, which addresses the two key challenges: (1) the compute intensity of PIR workloads, and (2) the need for scalable coordination across multiple computational storage devices (CSDs). We first present a resource-efficient hardware engine optimized for FHE computations, which maps the full PIR pipeline onto FPGA logic with minimal resource overhead (§ 6.1). We then introduce a load-aware task allocation strategy that balances PIR workloads across multiple CSDs to ensure system-level scalability (§ 6.2).

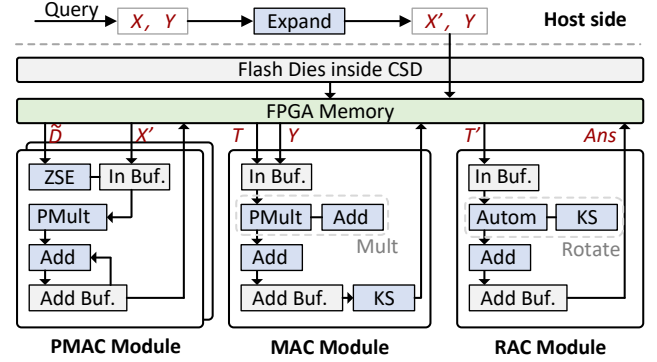


Figure 6: SmartPIR Engine microarchitecture.

6.1 SmartPIR Engine Design

FPGA-based acceleration for FHE (e.g., BFV) operations has been explored in prior work [2, 47, 63], usually leveraging pipelining and loop unrolling to improve throughput. However, these designs typically assume high-end FPGA cards with enough resources, whereas the FPGAs in CSDs impose much tighter resource constraints. To address this gap, we introduce a set of architectural optimizations that balance performance with resource efficiency. Specifically, our design (1) respects the limited logic and memory capacity of CSDs, (2) reduces the invocation frequency of costly BFV operations, and (3) calibrates pipeline depth to match realistic on-chip parallelism.

6.1.1 Resource Reuse. A Key challenge in designing the high-performance PIR engine lies in the large number of compute-intensive FHE operations—such as Add, PMult, and Mult—that must be executed over large datasets. A straightforward parallelization of these operations would rapidly exhaust FPGA resources due to duplication of sub-circuits and lack of temporal locality in execution. To address this, we first adopt a serial modular execution strategy that partitions the computation pipeline into three sequential stages—PMAC, MAC, RAC—based on functionality and dataflow patterns, as shown in Figure 6. This decomposition aligns with the progressive reduction in input scale across the execution process. Then, by leveraging the structural similarity among different FHE operations, we further selectively reuse partial hardware sub-modules. Specifically, we identify a set of core FHE operations needed across all modules, i.e., Add, PMult, and KeySwitch. We implement them as shared hardware blocks on the FPGA to alleviate resource pressure (e.g., BRAM, LUT, DSP), ensuring that each operation is used in a time-multiplexed fashion by whichever module currently requires it.

6.1.2 KeySwitch Hoisting. In the MAC module, ciphertext multiplication (i.e., Mult) inherently increases ciphertext dimensionality, necessitating a subsequent KeySwitch operation to project the ciphertext back into a manageable dimension. However, KeySwitch is expensive [25] and can increase the execution time of the MAC module by 1~2 orders of magnitude, making it a critical bottleneck. To this end, we adopt the *KeySwitch Hoisting* strategy to optimize both performance and resource utilization. Specifically, instead of performing a KeySwitch immediately after each Mult, it accumulates the intermediate products using repeated Add operations. The

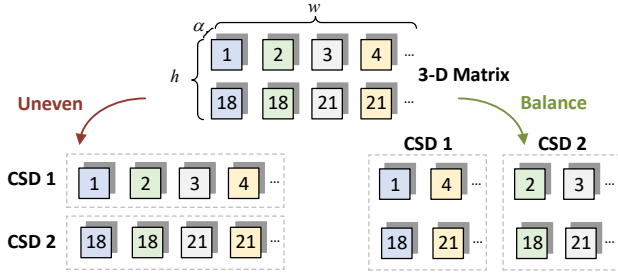


Figure 7: Data placement for PMAC. The value, e.g., 1,2, refers to the number of non-zero plaintexts in a group.

KeySwitch step is deferred until all multiplications in the current batch are completed (i.e., at $T'[\alpha]$ in Algorithm 2), as illustrated in Figure 6. This deferral significantly reduces the total number of KeySwitch operations. From a hardware scheduling perspective, fewer KeySwitch invocations simplify control logic and improve timing, as the MAC module can operate with a shallower and more predictable execution pipeline.

6.1.3 Design Space Explore. The three modules within the PIR engine contribute asymmetrically to overall system throughput, owing to differences in input dimensionality and computational complexity. Among them, PMAC is the primary bottleneck. As shown in Algorithm 2, the computational complexities of PMAC, MAC, and RAC are $O(\beta \cdot \alpha)$, $O(\sqrt{\beta/N} \cdot \alpha)$, and $O(\alpha)$, respectively. Consequently, the number of BFV operations in PMAC exceeds those in MAC and RAC by 3 to 4 orders of magnitude (which is further validated by the results presented in Table 4 in § 8.2). Therefore, to achieve an optimal balance between throughput and resource usage, SmartPIR employs a *module-aware* design space exploration. Specifically, we tailor the degree of parallelism in each sub-module to reflect its computational demands: for the PMAC module, its internal sub-modules are assigned a high degree of parallelism, and for MAC and RAC modules, a low degree of parallelism. In this way, we ensure that FPGA resources, particularly BRAM and DSP slices, are concentrated where they yield the largest performance gains. Meanwhile, less intensive stages are kept comparatively lean, mitigating intra-chip contention.

In summary, as shown in Figure 6, SmartPIR offloads all database-centric homomorphic operations, including Encode (ZSE), Add, PMult, Mult, and Rotate, to the CSDs. These operations are supported by reusable hardware circuits shared across the PMAC, MAC, and RAC, with computational complexity ranging from $O(N)$ to $O(N \log N)$ [51]. In contrast, query preprocessing operations such as Expand [7] are executed on the host, as they are computationally lightweight, involve no database access, and are intentionally kept off the CSD to avoid contention with stages such as PMAC.

6.2 Load-Aware Scheduling

As mentioned earlier, a hallmark of the SmartPIR architecture is its ability to scale both bandwidth and compute throughput linearly with the number of CSDs. However, to fully realize this potential, careful orchestration of data placement and task assignment is essential to avoid performance bottlenecks caused by workload skewness. With support for variable-length data, the computational

Table 1: Resource Utilization of FPGA Components.

	LUT	FF	BRAM	URAM	DSP
Count	357,869	390,185	840	80	904
Percentage	68.46%	37.32%	85.36%	62.50%	45.93%

load across partitions in SmartPIR becomes inherently imbalanced. This renders traditional even-split strategies ineffective. To address this, we design a load-aware scheduler that handles data distribution for PMAC and task execution for MAC and RAC, ensuring balanced and scalable performance across whole resources.

Data Placement for PMAC. The key idea behind data allocation is to achieve an even distribution of non-zero plaintexts across CSDs during system initialization, ensuring that each device handles a comparable local computational load during the PMAC phase. Since the server has clear knowledge of the number of non-zero plaintexts within each group of the matrix, the problem is modeled as a *classic knapsack problem* [9] and solved via a greedy heuristic. Specifically, the server iteratively assigns each plaintext group to the CSDs with the current lowest load, continuing until all plaintexts are distributed. To further refine this process, we introduce two system-level optimizations: (1) Since all plaintext groups within a matrix column share the same X' query in Algorithm 2, we treat each column as a single scheduling unit. This reduces scheduling granularity and enables point-to-point dispatch, avoiding broadcast overheads. (2) The column-wise plaintext counts, due to the sorting in ZSE, are monotonically increasing. Leveraging this structure, we adopt a sliding window mechanism with a width equal to the number of CSDs, enabling batch allocation that smooths workload variance. This hybrid strategy, as in Figure 7, achieves better balance than round-robin sharding.

Task Scheduling for MAC and RAC. In contrast to PMAC, scheduling for MAC and RAC prioritizes architectural simplicity over aggressive load balancing. Two insights motivate this design: (1) MAC and RAC computations rely on PMAC results and introduce synchronization and data exchange overheads if distributed. (2) The computational cost of MAC and RAC is substantially lower, by up to four orders of magnitude compared to PMAC (see § 6.1), making optimization less impactful on end-to-end performance. Given these considerations, we adopt a fully local execution model: each CSD independently performs its MAC and RAC modules without inter-device coordination or result merging. While this approach introduces modest computational redundancy, it drastically reduces architectural complexity and ensures seamless horizontal scaling, allowing CSDs to be added or removed without modifying the control plane or task management logic.

7 Implementation

Software Stack. SmartPIR is built atop the Microsoft SEAL library [51], leveraging the BFV scheme for homomorphic encryption. We choose the widely used FHT algorithm BFV in our implementation. To promise 128-bit security, we configure the BFV parameters as $(N, T, \log_2 Q) = (2^{13}, 2^{16} + 1, 218)$. The protocol requires a multiplicative depth of 2, eliminating the need for expensive Bootstrapping [5] and reducing cryptographic overhead.

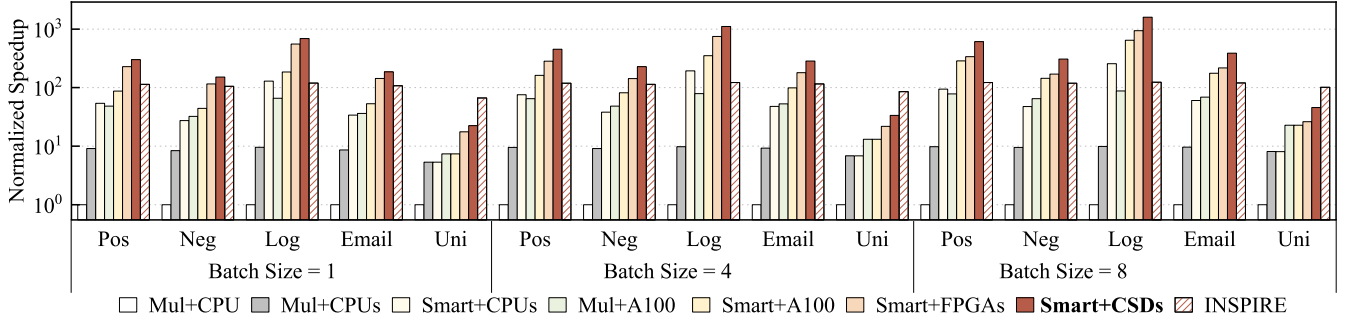


Figure 8: The performance comparison of different configurations. Here, Mul and Smart represent the MulPIR and SmartPIR protocols, and all results are normalized to the performance of MulPIR+CPU.

Hardware Integration. We adopt the commercial off-the-shelf SmartSSD [30] as the CSD in our work. SmartPIR’s runtime leverages the Xilinx OpenCL extension [41] to orchestrate host-FPGA interaction. During system initialization, the host constructs a routing table that maps encrypted X queries and database partitions to specific SmartSSD device IDs. This ensures precise offloading of each query to its designated compute/storage endpoint. Memory buffers are pre-allocated using the `CL_MEM_EXT_PTR_XILINX` flag, matching the maximum buffer capacity per device. These buffers are reused throughout the execution pipeline to store plaintexts from the 3D matrix, expanded queries X' , responses Y , and intermediate results T/T' . Data is transferred between the flash and FPGA through peer-to-peer (P2P) DMA, implemented via standard Linux `pread/pwrite` system calls. All computational logic—covering the PMAC, MAC, and RAC modules—is instantiated using Vitis HLS Tools. Table 1 reports the total FPGA resource utilization. Due to DRAM capacity limits on SmartSSDs, plaintexts are processed in successive batches, maintaining a steady streaming model between DRAM and the accelerator throughout query execution.

8 Evaluation

8.1 Experimental Setup

Our experiments are conducted on a server equipped with an Intel(R) Xeon(TM) Gold 5317 CPU @ 3.00GHz, 256 GB of DRAM, and running the Ubuntu 20.04 LTS operating system, with up to 6 Samsung SmartSSDs as computational storage devices. Each SmartSSD integrates a Kintex UltraScale+ KU15P FPGA, 4 GB of DDR4 DRAM, and 4 TB of NAND Flash, and they are connected via a PCIe expansion into the system. To establish the hardware baselines, we compare SmartPIR with both CPU and GPU configurations.

Baselines. We use MulPIR [6], which represents the state-of-the-art BFV-based PIR protocol as the software baseline. For performance comparison, we use the following configurations: (1) **MulPIR+CPU**, executed with a single thread on an Intel Xeon Gold 5317; (2) **MulPIR+CPUs** and (3) **SmartPIR+CPUs**, both deployed on the same CPU platform with 48 threads enabled; (4) **MulPIR+A100** and (5) **SmartPIR+A100**, utilizing the HEonGPU library on an NVIDIA A100 GPU; (6) **SmartPIR+FPGAs** follows

Table 2: Dataset Workloads (M=million).

	Pos	Neg	Log	Email	Uni
Entries Num.	120M	120M	800M	256M	67M
Database Size	0.2 TB	0.2 TB	0.35 TB	0.5 TB	1.0 TB

the design idea from FAST [61], a state-of-the-art FPGA FHE accelerator; and (7) **SmartPIR+CSDs**, integrates all our specific optimizations in § 6.1. Notably, both SmartPIR+FPGAs and SmartPIR+CSDs utilize 6 SmartSSDs as the storage devices. In the CPU or GPU-based configurations, the NVMe flash inside the SmartSSDs is accessed through a software RAID configuration built with Linux `mdadm` [11]. All datasets are partitioned across storage devices following the placement strategy described in § 6.2. In addition, we also compare our results with those of (8) **INSPIRE** [34], an in-storage PIR accelerator based on a simulated hardware architecture, using the performance results reported in their original paper. To ensure a fair comparison, we adopt the same BFV parameter set, $(N, T, \log_2 Q) = (2^{13}, 2^{16} + 1, 218)$, which is sufficient to support the multiplicative depth ($d = 2$) of all three PIR schemes, while still achieving the 128-bit security level (λ) [51].

Datasets. In our experiments, we evaluate the above configurations using four real-world, variable-length datasets, i.e., *IBDM-Pos* [23], *IBDM-Neg* [23], *Log* [67], and *Enron Email* [52]. Additionally, to assess the worst-case performance of SmartPIR, we synthesize a dataset of uniform length, referred to as *Uni*. More details of these datasets are provided in Table 2.

Metrics. We mainly focus on two primary performance metrics: (1) **Query Latency**, defined as the server-side time to process a query and produce the encrypted response; (2) **Communication Overhead**, measured as the total data communication (i.e., query and response) between client and server during query execution.

8.2 Performance Analysis

In this section, we compare the end-to-end performance of different configurations under different batch sizes and datasets, focusing on query latency and communication cost. For each test, a set of private queries is issued from the client, and the average performance per query is reported. Note that the server workload is independent of the client’s query, the server latency is the same for each query.

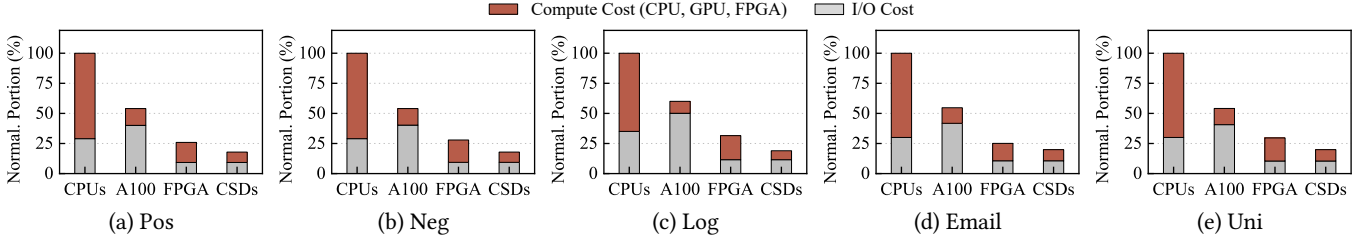


Figure 9: Time breakdown of SmartPIR deployed on CPUs, GPU, and CSDs (Batch Size = 1).

Table 3: Communication Overhead per Query.

	Pos	Neg	Log	Email	Uni
MulPIR	2.5 MB	1.8 MB	3.2 MB	3.2 MB	1.8 MB
SmartPIR	2.5 MB	2.5 MB	5.0 MB	3.2 MB	1.8 MB
INSPIRE	3.2 MB	3.2 MB	3.2 MB	3.2 MB	3.2 MB

Table 4: BFV Operations per Query on *Log* and *Uni* Datasets.

	MulPIR / SmartPIR(Log)	MulPIR / SmartPIR(Uni)
PMult	266.9M / 6.9M	67.1M / 67.1M
Add	266.9M / 6.9M	67.1M / 67.1M
Mult	16.3K / 6.4K	8.2K / 8.2K
Rotate	0 / 2.1K	0 / 8.2K

8.2.1 Overall Performance. Figure 8 presents the results of SmartPIR and MulPIR schemes across five datasets and three batch sizes. Clearly, SmartPIR+CSDs consistently achieves 1~3 orders of magnitude lower latency than all MulPIR configurations. Specifically, compared to MulPIR+CPU, SmartPIR+CSDs achieves average speedups of 455×, 229×, 1192×, 285×, and 34× on the *Neg*, *Pos*, *Log*, *Email*, and *Uni* datasets, respectively. When compared with MulPIR+CPUs, the corresponding speedups are decreased to a certain extent, which are 47×, 25×, 115×, 30×, and 4.9×, respectively. Even against the MulPIR+A100, SmartPIR+CSDs delivers notable improvements of 7.4×, 4.8×, 14.6×, 5.5×, and 2.4× across the same datasets. These performance gains arise from two main factors: (1) The in-storage processing design minimizes host-side I/O and parallelizes PIR execution across multiple SmartSSDs (detailed in § 8.2.2). (2) SmartPIR eliminates padding-related overhead by ZSE.

We further compare the performance of SmartPIR+CSDs with INSPIRE. Figure 8 shows that, on variable-length datasets, SmartPIR+CSDs consistently outperforms INSPIRE with average speedups of 4.8×, 14.6×, 5.5×, and 2.4× across the workloads *Pos*, *Neg*, *Log*, and *Email*, respectively. For the *Uni* dataset, the roles reverse, and INSPIRE is faster. Notably, the apparent advantage of INSPIRE rests on its software-simulated platform, which assumes idealized compute and I/O capabilities, thus masking its higher computational complexity relative to SmartPIR (even on the *Uni* dataset). In contrast, SmartPIR is fully implemented on commodity computational storage devices, offering better generality and reliability under realistic deployment conditions.

In terms of communication overhead, as illustrated in Table 3, the three schemes exhibit broadly similar communication costs across datasets. The small differences arise from their communication complexities: MulPIR benefits from packing m entries per plaintext, achieving $O(\sqrt{n/m}/N)$, while SmartPIR incurs $O(\sqrt{n}/N)$, leading to slightly higher overhead when entry sizes are small. INSPIRE, by contrast, employs hierarchical filtering to maintain a fixed query size regardless of dataset scale. Although these differences exist, they are minor relative to the dominant server-side latency and have a limited impact on overall end-to-end performance.

8.2.2 Breakdown Analysis. We further analyze the benefit gains of SmartPIR from both protocol and hardware perspectives.

Protocol-level gains. As shown in Figure 8, SmartPIR+CPUs achieves average speedups of 12.0×, 6.1×, 35.4×, 7.8×, and 1.0× over MulPIR+CPUs on *Neg*, *Pos*, *Log*, *Email*, and *Uni* datasets, respectively. These gains primarily stem from eliminating the overhead of padding. Unlike MulPIR, which standardizes all entries to a fixed length via padding, SmartPIR operates directly on variable-length payloads, reducing the number of plaintexts and, in turn, the number of required BFV operations. Table 4 highlights this reduction in the *Log* dataset, where SmartPIR performs significantly fewer PMult and Add operations. In contrast, for the *Uni* dataset, both systems perform an equivalent number of BFV operations, resulting in nearly identical performance. In addition, Table 4 also presents the dominant role of the PMAC stage in the overall system cost, as analyzed in § 6.1. This dominance stems from the overwhelming number of PMult operations — primarily used in PMAC—which is called far more frequently than Mult and Rotate operations employed in the MAC and RAC modules, respectively. Consequently, the PMAC stage accounts for over 98% of the total execution time.

Hardware-level gains. As shown in Figure 8, SmartPIR+CSDs achieves average speedup of 5.27×~5.69× over SmartPIR+CPUs; 2.41×~2.93× over SmartPIR+A100; and 1.55×~1.69× over SmartPIR+FPGAs across the five databases. These gains stem from two primary factors: (1) *Elimination of I/O bottlenecks.* Unlike CPU/GPU-based deployments, where storage and compute resources share PCIe buses, forcing SSDs to compete with each other for limited host bandwidth, SmartSSDs integrate compute capabilities directly into the drive and connect via dedicated PCIe lanes. By bypassing the host I/O stack, each SmartSSD delivers its own compute and bandwidth, so performance scales linearly as more devices are added. This not only provides larger aggregated bandwidth (more than 2× in our configurations) but also eliminates I/O contentions, making it ideally suited for PIR workloads. (2) *Hardware-aware protocol integration.* SmartPIR tightly couples protocol logic with hardware execution. It employs custom circuit modules, globally optimized

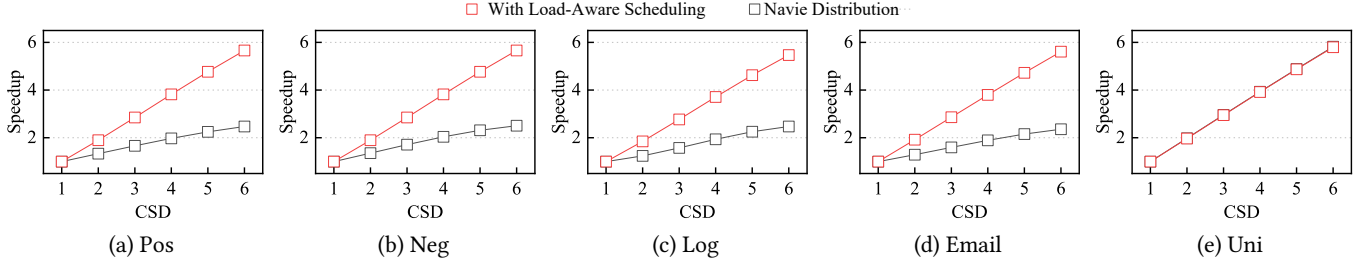


Figure 10: Scalability with CSDs of SmartPIR.

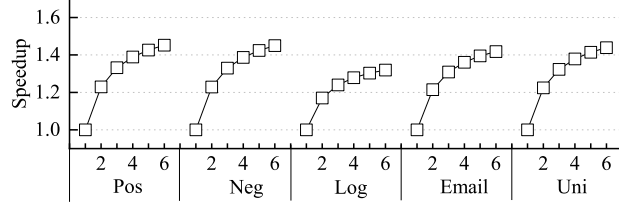


Figure 11: Scalability with A100 GPUs of SmartPIR.

parallelism, and fine-grained load balancing to maximize performance within the constraints of limited FPGA resources. Figure 9 presents the execution time breakdown for SmartPIR+CPUs, SmartPIR+A100, SmartPIR+FPGAs, and SmartPIR+CSDs. As shown, with 6 SmartSSDs, SmartPIR+CSDs significantly reduces I/O latency while maintaining compute performance comparable to SmartPIR+A100. Furthermore, compared to SmartPIR+FPGAs, our resource-efficient engine achieves an average 2.14 $\times$ compute speedup, demonstrating the effectiveness of our architectural design.

8.3 Scalability Analysis

We evaluate the scalability of SmartPIR+CSDs across 1 to 6 SmartSSDs, comparing our load-aware scheduling strategy with round-robin sharding [66]. In the round-robin sharding setup, entries are evenly partitioned across different SmartSSDs without accounting for variability in entry lengths. This results in a significant workload imbalance, causing certain devices to become bottlenecks and stalling the pipeline, thereby limiting system throughput. As shown in Figure 10, when handling variable-length datasets, performance gains under this naive strategy fall short of ideal linear scaling.

In contrast, our load-aware scheduler mitigates skewness by distributing entries based on estimated computational load using a sliding-window allocator and structure-aware grouping (§ 6.2). As illustrated in Figure 10, this approach enables SmartPIR to scale near-linearly across all datasets. When normalized to a single-device baseline, the system achieves average 1.91 $\times$, 3.81 $\times$, and 5.61 $\times$ speedups with 2, 4, and 6 SmartSSDs, respectively. Though our experiments involve a maximum number of 6 CSDs, theoretically, the performance gain of our design could be further improved with more devices. Two architectural advantages underpin this scalability: (1) Each SmartSSD contributes an independent FPGA compute unit and a dedicated PCIe uplink, enabling the system's aggregate compute and I/O bandwidth to scale linearly with the number of devices (§2.3). (2) Our load-aware scheduler ensures decentralized and balanced task distribution. These features allow

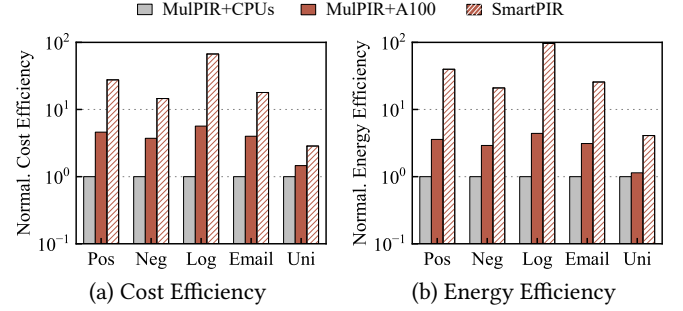


Figure 12: Cost and Energy efficiency of SmartPIR.

SmartPIR to exploit the parallelism introduced by additional CSDs fully, ensuring sustained performance gains as the system scales.

As a comparison, we also evaluate the horizontal scalability of the GPU-centric setting (i.e., SmartPIR+A100) by scaling the number of GPUs from 1 to 6. Figure 11 presents the resulting performance trends. While increasing to 6 A100 GPUs yields a near-linear 6 $\times$ improvement in peak GFLOPS, the actual performance gain over a single GPU remains below 1.5 $\times$. Meanwhile, its latency is approximately 2 $\times$ higher than that of the SmartPIR+CSDs configuration on average. This discrepancy is primarily due to the unchanged I/O bandwidth along the path from SSD to GPU via the CPU: when all 6 GPUs simultaneously handle the query, the limited host-side PCIe bandwidth still dominates the system efficiency.

8.4 Cost and Energy Efficiency

We also evaluate the cost and energy efficiency of three system configurations: MulPIR+CPUs, MulPIR+A100, and SmartPIR+CSDs. We report two key metrics: *performance/\$*, which captures throughput relative to system cost, and *joule-per-query*, which measures the energy required to complete a single query. Hardware prices are derived from current Amazon EC2 spot rates and include a shared server cost of \$15,000 and a storage cost of \$400 per SSD. For the latter two setups, additional hardware costs are estimated at \$7,000 for the A100 GPU and \$2,500 per SmartSSD. Energy measurements are obtained using a calibrated power analyzer.

As shown in Figure 12(a), despite the relatively high current market price of SmartSSDs, SmartPIR+CSDs consistently achieves higher cost efficiency than both MulPIR+CPUs and MulPIR+A100, with improvements ranging from 2.9 $\times$ to 67.2 $\times$ across the five datasets. These gains are primarily attributed to SmartPIR's significantly higher throughput, which effectively offsets the hardware

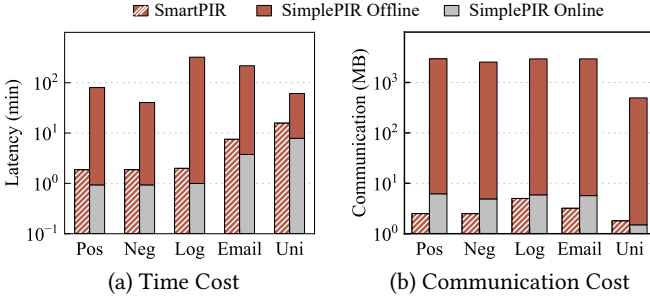


Figure 13: Evaluation of SmartPIR and SimplePIR.

cost compared to the CPUs/GPU-based baseline. A similar trend is observed in energy efficiency. As illustrated in Figure 12(b), SmartPIR+CSDs achieves energy savings ranging from $4.1\times$ to $96.6\times$ relative to the other two configurations. These improvements stem from both reduced query latency and the inherently lower power consumption of SmartSSDs, where embedded FPGAs consume substantially less power than high-end GPUs. Overall, SmartPIR demonstrates substantial advantages in both cost and energy efficiency, making it a practical and sustainable solution for large-scale, high-throughput deployments.

8.5 Comparison with LWE-based Scheme

To further demonstrate the efficiency and practicality of SmartPIR, we compare it with SimplePIR[26], a representative LWE-based PIR scheme that builds on the Regev encryption scheme[45]. SimplePIR achieves strong online performance by maintaining a large client-side state ($hint_c$) and performing extensive offline computation, which together help reduce the latency of real-time queries. However, this performance comes at a cost: SimplePIR exhibits poor support for dynamic datasets. Any update to the database necessitates re-executing the costly offline phase and re-synchronizing the client-side hint. We evaluate both protocols using the five datasets introduced in earlier sections.

Query Latency. SimplePIR divides query execution into an *offline phase* for heavy preprocessing and an *online phase* for light-weight query resolution. As shown in Figure 13(a), its online latency ranges from 0.9~7.9 minutes, slightly lower than SmartPIR’s 1.8~15.7 minutes. However, this modest gain is offset by its expensive offline phase, which takes 79~320 minutes and must be repeated after every database update—resulting in high amortized latency in dynamic scenarios. In contrast, SmartPIR avoids offline preprocessing, offering consistent query latency even under frequent updates.

Communication Overhead. SmartPIR incurs lower communication overhead, ranging from 1.8~5 MB per query across the four variable-length datasets. Even when excluding offline costs, SimplePIR’s online communication remains higher, ranging from 4.9~6.2 MB. On the uniform-length dataset, SimplePIR performs marginally better, though the difference is negligible. More critically, SimplePIR incurs substantial offline communication during updates, as refreshing $hint_c$ requires transferring 0.5~3.0 GB of data per update (see Figure 13(b)). In contrast, SmartPIR avoids offline synchronization, ensuring stable communication overhead.

9 Related Work

PIR Protocols. The performance of Private Information Retrieval (PIR) has improved markedly in recent years [4, 7, 22, 37, 39, 40], largely due to advances in RLWE-based Fully Homomorphic Encryption (FHE). These protocols enable private queries over public databases with strong security guarantees and form the foundation of most Index PIR systems. Beyond traditional index-based retrieval, several recent systems aim to support richer query semantics. Protocols such as Pantheon [3], CwPIR [36], PIRANA [35], SparsePIR [44], and Femur [65] extend PIR to support private key-value lookups and two-tuple queries. However, despite broadening the interface, these protocols still follow the uniform-length constraint, making them inefficient when applied to realistic datasets with variable-sized records. Concurrently, a complementary line of work explores sublinear-time PIR protocols under LWE-based cryptosystems. Recent schemes, such as Spiral [38], Hintless [31], WhisPIR [16], and SimplePIR [26] amortize expensive operations via extensive offline preprocessing to reduce online latency. While effective in static settings, these schemes often require large client-side state, limiting their practicality in dynamic or resource-constrained environments. In contrast, SmartPIR addresses the inefficiency of handling variable-length data and is designed to operate efficiently in dynamic datasets, without relying on offline preprocessing.

Hardware Acceleration. Fully Homomorphic Encryption (FHE) is the core cryptographic primitive in many PIR systems. Accordingly, a large body of work has focused on accelerating FHE operations to improve execution performance. These efforts leverage specialized hardware—including FPGAs [2, 47, 61, 63], GPUs [21, 43, 54, 60], and ASICs [17, 29, 34, 49, 50]—to exploit parallelism and custom circuits, achieving substantial reductions in computation time and increasing system throughput. However, these approaches largely focus on computation acceleration while overlooking the growing cost of data movement, which has become the dominant bottleneck in practical PIR deployments. Recent efforts have explored leveraging Processing-in-Memory (PIM) architectures to accelerate FHE-based Private Information Retrieval (PIR), aiming to alleviate the performance bottlenecks associated with frequent data movement. While promising in concept, most of these approaches [12, 19, 53, 68] are evaluated primarily through simulation, which makes it difficult to fully assess practical trade-offs in performance, area, and power consumption. Others [24, 42] are implemented on commercial platforms such as UPMEM [1], but have so far demonstrated only moderate performance gains in practice. In contrast, SmartPIR adopts the in-storage processing (ISP) paradigm to bridge this gap, offloading FHE computation directly to the storage layer. Built entirely on commercial off-the-shelf hardware, SmartPIR provides a deployable solution that delivers end-to-end performance and energy efficiency improvements.

10 Conclusion

In this paper, we present SmartPIR, a scalable PIR system that addresses key system-level bottlenecks during FHE-based private query processing. SmartPIR eliminates I/O overhead by offloading FHE operations to computational storage devices, and reduces computation via a zero-skipping encoding scheme that removes redundant processing incurred by variable-length data padding.

To fully realize the benefits of near-data processing, SmartPIR further integrates a resource-efficient FPGA design and a load-aware scheduling strategy for high-throughput and scalable execution. Experiments on a commercial CSD (i.e., SmartSSD) array demonstrate a $10^2 \times \sim 10^3 \times$ speedup over state-of-the-art CPU-based PIR schemes, validating the system's practicality for real-world deployments.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62432005, 62372272, 62272271, and 624B2091; in part by the Shandong Provincial Natural Science Foundation under Grants ZR2024MF099 and ZR2023LZH014; in part by the Department of Science and Technology of Shandong Province under Grant SYS202201; in part by the Quan Cheng Laboratory under Grant QCLZD202302; in part by the Taishan Scholars Program under Grant TSQN202211281; in part by the Open Research Program of the State Key Laboratory of Processors under Grant CLQ202405; and in part by the Qingmiao Program of Shandong university under Grant SDUQM2510.

References

- [1] 2025. UPMEM. <https://www.upmem.com/>.
- [2] Rashmi Agrawal, Leo de Castro, Guowei Yang, Chiraag Juvekar, Rabia Yazicigil, Anantha Chandrakasan, Vinod Vaikuntanathan, and Ajay Joshi. 2023. FAB: An FPGA-based accelerator for bootstrappable fully homomorphic encryption. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 882–895. <https://ieeexplore.ieee.org/abstract/document/10070953>.
- [3] Ishfaq Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2022. Pantheon: Private retrieval from public key-value store. *Proceedings of the VLDB Endowment* 16, 4 (2022), 643–656. <https://dl.acm.org/doi/pdf/10.14778/3574245.3574251>.
- [4] Ishfaq Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addr: Metadata-private voice communication over fully untrusted infrastructure. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. <https://www.usenix.org/system/files/osdi21-ahmad.pdf>.
- [5] Ahmad Al Badawi and Yuriy Polyakov. 2023. Demystifying bootstrapping in fully homomorphic encryption. *Cryptology ePrint Archive* (2023). <https://eprint.iacr.org/2023/149.pdf>.
- [6] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Philipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. Communication-Computation Trade-offs in PIR. In *30th USENIX Security Symposium (Security)*. 1811–1828. <https://www.usenix.org/system/files/sec21-ali.pdf>.
- [7] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with compressed queries and amortized query processing. In *IEEE Symposium on Security and Privacy (SP)*. 962–979. <https://eprint.iacr.org/2017/1142.pdf>.
- [8] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. 2012. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*. 53–64. <https://dl.acm.org/doi/pdf/10.1145/2254756.2254766>.
- [9] Valentina Cacchiani, Manuel Iori, Alberto Locatelli, and Silvano Martello. 2022. Knapsack problems—An overview of recent advances. Part II: Multiple, multi-dimensional, and quadratic knapsack problems. *Computers & Operations Research* 143 (2022), 105693. <https://www.sciencedirect.com/science/article/pii/S0305054821003889>.
- [10] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. 2020. Characterizing, modeling, and benchmarking RocksDB Key-Value workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST)*. 209–223. https://www.usenix.org/system/files/fast20-cao_zhichao.pdf.
- [11] Peter M Chen, Edward K Lee, Garth A Gibson, Randy H Katz, and David A Patterson. 1994. RAID: High-performance, reliable secondary storage. *ACM Computing Surveys (CSUR)* 26, 2 (1994), 145–185. <https://dl.acm.org/doi/pdf/10.1145/176979.176981>.
- [12] Zehao Chen, Zhining Cao, Zhaoyan Shen, and Lei Ju. 2024. HMC-FHE: A Heterogeneous Near Data Processing Framework for Homomorphic Encryption. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 43, 11 (2024), 3551–3563. <https://ieeexplore.ieee.org/abstract/document/10745817>.
- [13] Jung Hee Cheon, Hyeonmin Choe, Alain Passelègue, Damien Stehlé, and Elias Suvanto. 2024. Attacks against the IND-CPAD security of exact FHE schemes. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2505–2519. <https://dl.acm.org/doi/pdf/10.1145/3658644.3690341>.
- [14] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. 1998. Private information retrieval. *Journal of the ACM (JACM)* 45, 6 (1998), 965–981. <https://dl.acm.org/doi/pdf/10.1145/293347.293350>.
- [15] James C Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, Jeffrey John Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, et al. 2013. Spanner: Google's globally distributed database. *ACM Transactions on Computer Systems* 31, 3 (2013), 1–22. <https://dl.acm.org/doi/pdf/10.1145/2491245>.
- [16] Leo de Castro, Kevin Lewi, and Edward Suh. 2024. WhisPIR: Stateless Private Information Retrieval with Low Communication. *Cryptology ePrint Archive* (2024). <https://eprint.iacr.org/2024/270.pdf>.
- [17] Xianglong Deng, Shengyu Fan, Zhicheng Hu, Zhuoyu Tian, Zihao Yang, Jiangrui Yu, Dingyuan Cao, Dan Meng, Rui Hou, Meng Li, et al. 2024. Trinity: A General Purpose FHE Accelerator. In *2024 57th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 338–351. <https://arxiv.org/pdf/2410.13405>.
- [18] Tharam Dillon, Chen Wu, and Elizabeth Chang. 2010. Cloud computing: issues and challenges. In *224th IEEE International Conference on Advanced Information Networking and Applications (AINA)*. 27–33. <https://ieeexplore.ieee.org/abstract/document/5474674>.
- [19] Lin Ding, Song Bian, Penggao He, Yan Xu, Gang Qu, and Jiliang Zhang. 2024. APACHE: A Processing-Near-Memory Architecture for Multi-Scheme Fully Homomorphic Encryption. *arXiv preprint arXiv:2404.15819* (2024). <https://arxiv.org/pdf/2404.15819>.
- [20] Zhuohui Duan, Hao Feng, Haikun Liu, Xiaofei Liao, Hai Jin, and Bangyu Li. 2025. {AegonKV}: A High Bandwidth, Low Tail Latency, and Low Storage Cost {KV-Separated} {LSM} Store with {SmartSSD-based} {GC} Offloading. In *23rd IEEE Conference on File and Storage Technologies (FAST)*. 321–335. <https://www.usenix.org/system/files/fast25-duan.pdf>.
- [21] Shengyu Fan, Zhiwei Wang, Weizhi Xu, Rui Hou, Dan Meng, and Mingzhe Zhang. 2023. Tensorfhe: Achieving practical computation on encrypted data using gpgpu. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 922–934. <https://ieeexplore.ieee.org/abstract/document/10071017>.
- [22] Ben Fisch, Arthur Lazzaretti, Zeyu Liu, and Charalampos Papamanthou. 2024. ThorPIR: single server PIR via homomorphic thorp shuffles. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1448–1462. <https://dl.acm.org/doi/pdf/10.1145/3658644.3690326>.
- [23] Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2020. The pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027* (2020). <https://arxiv.org/pdf/2101.00027>.
- [24] Harshita Gupta, Mayank Kabra, Juan Gómez-Luna, Konstantinos Kanellopoulos, and Onur Mutlu. 2023. Evaluating homomorphic operations on a real-world processing-in-memory system. In *2023 IEEE International Symposium on Workload Characterization (IISWC)*. 211–215. <https://ieeexplore.ieee.org/abstract/document/10289253>.
- [25] Shai Halevi, Yuriy Polyakov, and Victor Shoup. 2019. An improved RNS variant of the BFV homomorphic encryption scheme. In *The Cryptographer's Track at RSA Conference (CT-RSA)*. 83–105. <https://eprint.iacr.org/2018/117.pdf>.
- [26] Alexandra Henzinger, Matthew M Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. 2023. One server for the price of two: Simple and fast {Single-Server} private information retrieval. In *32nd USENIX Security Symposium (USENIX Security)*. 3889–3905. <https://www.usenix.org/system/files/usenixsecurity23-henzinger.pdf>.
- [27] Hongsun Jang, Jaeyong Song, Jaewon Jung, Jaeyoung Park, Youngsok Kim, and Jinho Lee. 2024. Smart-Infinity: Fast Large Language Model Training using Near-Storage Processing on a Real System. In *2024 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 345–360. <https://ieeexplore.ieee.org/abstract/document/10476401>.
- [28] Seongyoung Kang and Sang-Woo Jun. 2024. Sting: Near-storage accelerator framework for scalable triangle counting and beyond. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*. 1–6. <https://dl.acm.org/doi/pdf/10.1145/3649329.3658265>.
- [29] Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, et al. 2022. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 711–725. <https://dl.acm.org/doi/pdf/10.1145/3470496.3527393>.
- [30] Joo Hwan Lee, Hui Zhang, Veronica Lagrange, Praveen Krishnamoorthy, Xiaodong Zhao, and Yang Seok Ki. 2020. SmartSSD: FPGA accelerated near-storage data analytics on SSD. *IEEE Computer architecture letters* 19, 2 (2020), 110–113. <https://ieeexplore.ieee.org/abstract/document/9141369>.
- [31] Baiyu Li, Daniele Micciancio, Mariana Raykova, and Mark Schultz-Wu. 2024. Hintless single-server private information retrieval. In *Annual International Cryptology Conference (CRYPTO)*. 183–217. https://link.springer.com/chapter/10.1007/978-3-031-68400-5_6.
- [32] Feifei Li. 2019. Cloud-native database systems at Alibaba: Opportunities and challenges. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2263–2272. <https://>

- //vldb.org/pvldb/vol12/p2263-li.pdf.
- [33] Guoliang Li, Wengang Tian, Jinyu Zhang, Ronen Grosman, Zongchao Liu, and Sihao Li. 2024. GaussDB: A Cloud-Native Multi-Primary Database with Compute-Memory-Storage Disaggregation. *Proceedings of the VLDB Endowment* 17, 12 (2024), 3786–3798. <https://dl.acm.org/doi/pdf/10.14778/3685800.3685806>.
 - [34] Jilan Lin, Ling Liang, Zheng Qu, Ishtiyaque Ahmad, Liu Liu, Fengbin Tu, Trinabh Gupta, Yufei Ding, and Yuan Xie. 2022. INSPIRE: in-storage private information retrieval via protocol and architecture co-design. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 102–115. <https://dl.acm.org/doi/pdf/10.1145/3470496.3527433>.
 - [35] Jian Liu, Jingyu Li, Di Wu, and Kui Ren. 2024. PIRANA: Faster multi-query PIR via constant-weight codes. In *2024 IEEE Symposium on Security and Privacy (SP)*. 4315–4330. <https://ieeexplore.ieee.org/abstract/document/10646690>.
 - [36] Rasoul Akhavan Mahdavi and Florian Kerschbaum. 2022. Constant-weight {PIR}: Single-round keyword PIR via constant-weight equality operators. In *31st USENIX Security Symposium (USENIX Security)*. 1723–1740. <https://www.usenix.org/system/files/sec22-mahdavi.pdf>.
 - [37] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies (PoPETs)* (2016), 155–174. <https://hal.science/hal-01396142/>.
 - [38] Samir Jordan Menon and David J Wu. 2022. Spiral: Fast, high-rate single-server PIR via FHE composition. In *2022 IEEE Symposium on Security and Privacy (SP)*. 930–947. <https://ieeexplore.ieee.org/abstract/document/9833700>.
 - [39] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response efficient single-server PIR. In *Proceedings of the ACM SIGSAC conference on computer and communications security (CCS)*. 2292–2306. <https://dl.acm.org/doi/pdf/10.1145/3460120.3485381>.
 - [40] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy (SP)*. 437–452. <https://par.nsf.gov/servlets/purl/10539487>.
 - [41] Aaftab Munshi. 2009. The openssl specification. In *2009 IEEE Hot Chips 21 Symposium (HCS)*. 1–314. <https://ieeexplore.ieee.org/abstract/document/7478342>.
 - [42] Mpoki Mwaisela, Joel Hari, Peterson Yuhala, James Ménétrey, Pascal Felber, and Valerio Schiavoni. 2024. Evaluating the Potential of In-Memory Processing to Accelerate Homomorphic Encryption: Practical Experience Report. In *2024 43rd International Symposium on Reliable Distributed Systems (SRDS)*. 92–103. <https://ieeexplore.ieee.org/abstract/document/10806506>.
 - [43] Ali Şah Özcan and Erkay Savaş. 2024. HEonGPU: a GPU-based Fully Homomorphic Encryption Library 1.0. *Cryptology ePrint Archive* (2024). <https://eprint.iacr.org/2024/1543.pdf>.
 - [44] Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. {Don't} be Dense: Efficient Keyword {PIR} for Sparse Databases. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3853–3870. <https://www.usenix.org/system/files/usenixsecurity23-patel.pdf>.
 - [45] Oded Regev. 2004. New lattice-based cryptographic constructions. *Journal of the ACM (JACM)* 51, 6 (2004), 899–942. <https://dl.acm.org/doi/abs/10.1145/1039488.1039490>.
 - [46] Michael K. Reiter and Avi D. Rubin. 1999. Anonymous Web Transactions with Crowds. *Commun. ACM* 42, 2 (1999), 32–38. <https://dl.acm.org/doi/pdf/10.1145/293411.293778>.
 - [47] M Sadeh Riazi, Kim Laine, Blake Pelton, and Wei Dai. 2020. HEAX: An architecture for computing on encrypted data. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLoS)*. 1295–1309. <https://dl.acm.org/doi/pdf/10.1145/3373376.3378523>.
 - [48] Miruna Rosca, Damien Stehlé, and Alexandre Wallet. 2018. On the ring-LWE and polynomial-LWE problems. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 146–173. <https://eprint.iacr.org/2018/170.pdf>.
 - [49] Nikola Samardžić, Axel Feldmann, Aleksandar Krastev, and et al. 2021. F1: A fast and programmable accelerator for fully homomorphic encryption. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 238–252. <https://dl.acm.org/doi/pdf/10.1145/3466752.3480070>.
 - [50] Nikola Samardžić, Axel Feldmann, Aleksandar Krastev, and et al. 2022. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*. 173–187. <https://dl.acm.org/doi/pdf/10.1145/3470496.3527393>.
 - [51] SEAL 2023. Microsoft SEAL (release 4.1). <https://github.com/Microsoft/SEAL>. Microsoft Research, Redmond, WA..
 - [52] Jitesh Shetty and Jafar Adibi. 2004. The Enron email dataset database schema and brief statistical report. *Information sciences institute technical report, University of Southern California* 4, 1 (2004), 120–128. https://foreverdata.org/1009HOLD/Enron_Dataset_Report.pdf.
 - [53] Shangyi Shi, Husheng Han, Jianan Mu, Xinyao Zheng, Ling Liang, Hang Lu, Zidong Du, Xiaowei Li, Xing Hu, and Qi Guo. 2025. FlexMem: High-Parallel Near-Memory Architecture for Flexible Dataflow in Fully Homomorphic Encryption. *arXiv preprint arXiv:2503.23496* (2025). <https://arxiv.org/pdf/2503.23496>.
 - [54] Kaustubh Shivdikar, Yuhui Bao, Rashmi Agrawal, Michael Shen, Gilbert Jonatan, Evelio Mora, Alexander Ingare, Neal Livesay, José L. Abellán, John Kim, et al. 2023. Gme: Gpu-based microarchitectural extensions to accelerate homomorphic encryption. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 670–684. <https://dl.acm.org/doi/pdf/10.1145/3613424.3614279>.
 - [55] Bing Tian, Haikun Liu, Zhuohui Duan, Xiaofei Liao, Hai Jin, and Yu Zhang. 2024. Scalable Billion-point Approximate Nearest Neighbor Search Using SmartSSDs. In *2024 USENIX Annual Technical Conference (USENIX ATC)*. 1135–1150. <https://www.usenix.org/system/files/atc24-tian.pdf>.
 - [56] Bing Tian, Haikun Liu, Yuhang Tang, Shihai Xiao, Zhuohui Duan, Xiaofei Liao, Hai Jin, Xuecang Zhang, Junhua Zhu, and Yu Zhang. 2025. Towards High-throughput and Low-latency Billion-scale Vector Search via {CPU/GPU} Collaborative Filtering and Re-ranking. In *23rd USENIX Conference on File and Storage Technologies (FAST)*. 171–185. <https://www.usenix.org/system/files/fast25-tian-bing.pdf>.
 - [57] Sennur Ulukus, Salman Avestimehr, Michael Gastpar, Syed A Jafar, Ravi Tandon, and Chao Tian. 2022. Private retrieval, computing, and learning: Recent progress and future challenges. *IEEE Journal on Selected Areas in Communications* 40, 3 (2022), 729–748. <https://ieeexplore.ieee.org/ielam/49/9715277/9678331-aam.pdf?tag=1>.
 - [58] Shengzhe Wang, Zihang Lin, Suzhen Wu, Hong Jiang, Jie Zhang, and Bo Mao. 2024. LearnedFTL: A Learning-Based Page-Level FTL for Reducing Double Reads in Flash-Based SSDs. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 616–629. <https://ieeexplore.ieee.org/abstract/document/10476463>.
 - [59] Yuyue Wang, Xiurui Pan, Yuda An, Jie Zhang, and Glenn Reinman. 2024. Beacon-GNN: Large-Scale GNN Acceleration with Out-of-Order Streaming In-Storage Computing. In *IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 330–344. <https://ieeexplore.ieee.org/abstract/document/10476427>.
 - [60] Zhiwei Wang, Haoqi He, Lutan Zhao, Peinan Li, Zhihao Li, Dan Meng, and Rui Hou. 2024. Chameleon: An Efficient FHE Scheme Switching Acceleration on GPUs. *arXiv preprint arXiv:2410.05934* (2024). <https://arxiv.org/pdf/2410.05934>.
 - [61] Zhihan Xu, Tian Ye, Rajgopal Kannan, and Viktor K Prasanna. 2025. FAST: FPGA Acceleration of Fully Homomorphic Encryption with Efficient Bootstrapping. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*. 115–126. <https://dl.acm.org/doi/pdf/10.1145/3706628.3708879>.
 - [62] Pan Yang, Naixue Xiong, and Jingli Ren. 2020. Data security and privacy protection for cloud storage: A survey. *IEEE Access* 8 (2020), 131723–131740. <https://ieeexplore.ieee.org/abstract/document/9142202>.
 - [63] Yinghao Yang, Huaizhi Zhang, Shengyu Fan, Hang Lu, Mingzhe Zhang, and Xiaowei Li. 2023. Poseidon: Practical homomorphic encryption accelerator. In *2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. 870–881. <https://ieeexplore.ieee.org/abstract/document/10070984>.
 - [64] Zhe Yang, Youyou Lu, Xiaojian Liao, Youmin Chen, Junru Li, Siyu He, and Jiwu Shu. 2023. {λ-IO}: A Unified {IO} Stack for Computational Storage. In *21st USENIX Conference on File and Storage Technologies (FAST)*. 347–362. <https://www.usenix.org/system/files/fast23-yang-zhe.pdf>.
 - [65] Jiaoyi Zhang, Liqiang Peng, Mo Sha, Weiran Liu, Xiang Li, Sheng Wang, Feifei Li, Mingyu Gao, and Huanchen Zhang. 2025. Femur: A Flexible Framework for Fast and Secure Querying from Public Key-Value Store. *arXiv preprint arXiv:2503.05376* (2025). <https://arxiv.org/pdf/2503.05376>.
 - [66] Yunjia Zheng, Yuxuan Tian, Joseph Vinish D'silva, and Bettina Kemme. 2023. DBMLSchd: Scheduling In-database Machine Learning Jobs.. In *VLDB Workshops*. <https://ceur-ws.org/Vol-3462/AIDB3.pdf>.
 - [67] Junwei Zhou, Yijia Qian, Qingtian Zou, Peng Liu, and Jianwen Xiang. 2022. Deepsyslog: Deep anomaly detection on syslog using sentence embedding and metadata. *IEEE Transactions on Information Forensics and Security* 17 (2022), 3051–3061. <https://ieeexplore.ieee.org/abstract/document/9865986/>.
 - [68] Minxuan Zhou, Yujin Nam, Pranav Gangwar, Weihong Xu, Arpan Dutta, Chris Wilkerson, Rosario Cammarota, Saransh Gupta, and Tajana Rosing. 2025. FHE-mem: A processing in-memory accelerator for fully homomorphic encryption. *IEEE Transactions on Emerging Topics in Computing* (2025). <https://ieeexplore.ieee.org/abstract/document/10845064>.