Femur: A Flexible Framework for Fast and Secure Querying from Public Key-Value Store

JIAOYI ZHANG, Tsinghua University, China
LIQIANG PENG, Alibaba Group, China
MO SHA, Alibaba Cloud, Singapore
WEIRAN LIU, Alibaba Group, China
XIANG LI, Tsinghua University, China
SHENG WANG, Alibaba Cloud, Singapore
FEIFEI LI, Alibaba Cloud, China
MINGYU GAO*, Tsinghua University, China
HUANCHEN ZHANG*, Tsinghua University, China

With increasing demands for privacy, it becomes necessary to protect sensitive user query data when accessing public key-value databases. Existing Private Information Retrieval (PIR) schemes provide full security but suffer from poor scalability, limiting their applicability in large-scale deployment. We argue that in many real-world scenarios, a more practical solution should allow users to flexibly determine the privacy levels of their queries in a theoretically guided way, balancing security and performance based on specific needs. To formally provide provable guarantees, we introduce a novel concept of distance-based indistinguishability, which can facilitate users to comfortably relax their security requirements. We then design Femur, an efficient framework to securely query public key-value stores with flexible security and performance trade-offs. It uses a space-efficient learned index to convert query keys into storage locations, obfuscates these locations with extra noise provably derived by the distance-based indistinguishability theory, and sends the expanded range to the server. The server then adaptively utilizes the best scheme to retrieve data. We also propose a novel variable-range PIR scheme optimized for bandwidth-constrained environments. Experiments show that Femur outperforms the state-of-the-art designs even when ensuring the same full security level. When users are willing to relax their privacy requirements, Femur can further improve the performance gains to up to 163.9×, demonstrating an effective trade-off between security and performance.

CCS Concepts: • **Security and privacy** → **Privacy-preserving protocols**.

Additional Key Words and Phrases: Public Key-Value Store, Private Information Retrieval

ACM Reference Format:
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. *Proc. ACM Manag. Data* 3, 3 (SIGMOD), Article 162 (June 2025), 29 pages. https://doi.org/10.1145/3725299

*Huanchen Zhang and Mingyu Gao are also affiliated with Shanghai Qi Zhi Institute. Corresponding authors.

Authors' Contact Information: Jiaoyi Zhang, jy-zhang20@mails.tsinghua.edu.cn, Tsinghua University, Beijing, China; Liqiang Peng, plq270998@alibaba-inc.com, Alibaba Group, Beijing, China; Mo Sha, shamo.sm@alibaba-inc.com, Alibaba Cloud, Singapore, Singapore; Weiran Liu, weiran.lwr@alibaba-inc.com, Alibaba Group, Beijing, China; Xiang Li, lixiang20@mails.tsinghua.edu.cn, Tsinghua University, Beijing, China; Sheng Wang, sh.wang@alibaba-inc.com, Alibaba Cloud, Singapore, Singapore; Feifei Li, lifeifei@alibaba-inc.com, Alibaba Cloud, Hangzhou, China; Mingyu Gao, gaomy@tsinghua.edu.cn, Tsinghua University, Beijing, China; Huanchen Zhang, huanchen@tsinghua.edu.cn, Tsinghua University, Beijing, China.

ACM 2836-6573/2025/6-ART162
https://doi.org/10.1145/3725299

1 Introduction

The growing data volumes and pervasive cloud services have spurred efforts to protect user data privacy. Protecting user queries to public datasets is as critical as protecting the underlying database. For example, users checking phone numbers against a public scam database are vested in keeping the numbers private to the hosting server. Similarly, users querying sensitive health or financial data risk potential discrimination (e.g., biased treatment in insurance or employment) if their queries are revealed. These use cases highlight the need for secure and efficient query mechanisms for accessing public key-value stores while keeping queries confidential.

Private Information Retrieval (PIR) schemes [28, 29, 45, 50, 59, 60, 64] have emerged to address the above concerns. They allow users to upload encrypted keys for confidential server-side computation and results return without decryption. The entire process ensures that sensitive information remains secure. Despite advancements in PIR techniques—such as improvements in cryptographic protocols [7, 23, 31, 42, 57], encoding strategies [13], and pre-processing optimizations [25, 27, 31, 95]—they still face challenges with scaling. To guarantee query path privacy, most designs must process all the n key-value pairs in the database with up to $O(n)$ complexity, leading to high query response time in large-scale applications. For example, Pantheon [7] takes 1.15 seconds to execute a query on only 65,536 key-value pairs.

In this paper, we propose a more practical solution that relaxes security while upholding theoretical guarantees. Existing PIR schemes yield impractical processing times for large real-world datasets, with delays ranging from seconds to hours (Pantheon [7] takes over 2 hours on 2^{24} records). Moreover, full dataset obfuscation may not be necessary in many applications. For example, an American user may only need to hide their queries within the U.S. phone numbers rather than the global dataset. In fact, industry practices often relax security by using hash functions [43] to partition datasets into smaller buckets to improve query performance by reducing the number of key-value pairs involved per query. However, this method lacks formal security guarantees: the bucket IDs are exposed during computation, and an uneven hash partition could produce single-pair buckets, thus revealing the user's query.

We introduce Femur, a framework enabling users to control and balance performance and privacy with theoretical security guarantees. Femur allows users to configure from no security to full security backed by our formal definition of "relaxed security" (i.e., *distance-based indistinguishability*). Depending on the chosen level, Femur's user-side module automatically sends an obfuscated range of keys (including the user's real key) to the server, ensuring that the real key is indistinguishable under the relaxed security level without unnecessary performance loss. When this key range extends to the entire database, Femur achieves full privacy as before.

Femur consists of three main components: key-to-position conversion, obfuscated range generation, and adaptive key-value retrieval. First, the user-side *key-to-position conversion* module maps query keys to their storage locations within the database, similar to existing keyword PIRs. The novelty lies in that we use the PGM-index [40], a state-of-the-art, space-efficient learned index for the mapping. The prediction error of PGM-index can be handled together with the obfuscated query range. Learned indexes are typically faster and smaller than traditional indexes (e.g., B+trees) and can therefore speed up the conversion while significantly reducing the index size stored on the client side.

Second, the *obfuscated range generation* module adds noise to the range predicted by the PGM-index before sending it to the server so that the server-side computation (or transmission) is confined to the key-value pairs within the range. This obfuscated range satisfies distance-based indistinguishability with a user-specified security level guaranteed by our theory. The higher the security level, the slower the query.

Third, the server-side *adaptive key-value retrieval* module employs a cost model to choose the best strategy, trading between computation and network bandwidth. For high-bandwidth networks, the client side downloads all key-value pairs in the obfuscated range (of length s) in plaintext. This approach conceals the query key without homomorphic computation but incurs $O(s)$ transmission complexity. When the bandwidth is limited, we propose a novel variable-range PIR scheme with a transmission complexity of $O(1)$ but a computation complexity of $O(s)$.

Our experiments show that Femur scales well on a dataset containing 200 million records (a size infeasible for previous solutions) and outperforms the two baselines (Chalamet [25] and Pantheon [7]) by 1.05× and 7.71×, respectively under full security, and by 163.9× and 1206.1×, respectively with a relaxed security guarantee. Additionally, Femur's offline initialization phase only takes a few minutes, compared to several hours from previous designs. We also integrated Femur into Redis [1], a popular in-memory key-value store, and demonstrated that Femur also supports efficient value updates in real-world scenarios.

We make the following contributions.

- We identify that full security prevents current PIR schemes from scaling and is often unnecessary in real-world applications.
- We introduce relaxed security by formally defining "distance-based indistinguishability" to allow users to trade between performance and privacy while offering provable security guarantees.
- We propose Femur, a framework that eliminates unnecessary computational and communication costs by incorporating a learned index with a novel variable-range PIR scheme.
- We demonstrate Femur's superior performance and scalability via a thorough evaluation.

2 Preliminaries

2.1 Private Retrieval From Public Data

This paper focuses on user privacy in public key-value stores by ensuring secure lookups that hide both the users' query keys and access patterns. This is often realized by traditional PIR schemes, including keyword PIRs and index PIRs. Depending on whether pre-processing is allowed, PIR schemes can be further categorized as stateless PIR (without pre-processing) [7, 8, 13, 59] often with linear computational complexity, and as stateful PIRs [25, 29, 31, 45, 50, 95] which leverage offline pre-processing to reduce online computation.

2.1.1 Index-PIR. Index PIRs assume that the user knows the exact location of the queried data. Typically, the user encodes the location as a one-hot vector, encrypts it, and sends it to an untrusted server. The server then performs a privacy-preserving inner product between the encrypted query and its dataset, returning the encrypted result to the user. We focus on the single-server scenario, as the multi-server setup relies on the less practical assumption of non-colluding servers.

Existing index-PIR schemes like SimplePIR [45] and FrodoPIR [31] reduce online computation via one-time pre-computation, where users download specific "hints" in the offline phase. However, their online phase still scales linearly with dataset size. Piano [95] takes a different approach where the user continuously uploads sets of multiple locations during pre-processing and obtains the XOR of these data as hints. The online query must match one of these hints, accelerating queries but incurring offline communication overhead equal to the dataset size, rendering it impractical. Since offline interactions between the user and the server are inevitable (e.g., establishing a communication channel), Femur also allows transferring hints during the offline phase. However, it ensures that the hint size remains significantly smaller than the dataset. To achieve this, Femur introduces a novel design that leverages learned indexes for faster pre-processing and efficient online processing.

Additionally, optimizations in cryptographic primitives [60, 64] and batch processing [65, 87] are orthogonal to our design and could enhance our framework with minor adjustments.

2.1.2 Keyword-PIR. In real-world scenarios, users often do not know the exact location of the data they wish to retrieve. Keyword PIRs address this limitation by allowing queries based on keywords, and they are typically implemented on top of index PIRs. The earliest approach [28] achieves this by mapping keywords to data locations through logarithmic rounds of communication. However, this introduces substantial overhead, rendering the method inefficient. For example, retrieving data from one million key-value pairs may require up to 21 round trips, resulting in significant overhead.

Recent keyword PIR schemes reduce rounds to one. Approaches based on fully homomorphic encryption (FHE) [23, 42] achieve this by using equality operators [7, 57]. In these schemes, the user encrypts the desired keyword and uploads it to the server, which performs an equality check between the encrypted keyword and all keys in the dataset. This results in an encrypted one-hot vector that can be used for consequent index PIR. While these methods reduce communication, equality checks remain computationally expensive. To further enhance efficiency, Chalamet [25] employs techniques such as cuckoo hashing or filters to map keys to multiple potential locations, using index PIR to retrieve and combine these values. Compared to FHE-based methods, Chalamet reduces the online server-side computations, but still incurs high communication overhead and requires substantial pre-processing.

These schemes improve retrieval efficiency while maintaining full security. However, Femur provides extra flexibility by allowing users to adjust security levels according to their specific needs, achieving a better balance between performance and privacy.

2.2 Differential Privacy

Differential privacy (DP) is a rigorous mathematical method widely used in database systems, aimed at preventing adversaries from inferring individual data from query results by adding controlled noise. Classical DP applications assume the presence of a trusted party that collects all the data and adds noise to the entire dataset, an approach adopted by various organizations including Google [12] and Uber [46, 47]. In contrast, Local Differential Privacy (LDP) removes the need for a trusted party by allowing users to obfuscate data locally before sharing, as used by Google [9, 36], Apple [77], and Meta [61]. Besides, users have the flexibility to choose their desired privacy level based on the sensitivity of their information [11].

Definition 1 (Local Differential Privacy, LDP). *An obfuscation mechanism $\mathcal{M} : \mathbb{D} \longrightarrow \mathbb{O}$ satisfies ϵ-LDP with privacy level ϵ ($\epsilon \geq 0$) if for any $x, x' \in \mathbb{D}$ and any output $y \in \mathbb{O}$, we have*

$$\Pr[\mathcal{M}(x) = y] \leq e^{\epsilon} \cdot \Pr[\mathcal{M}(x') = y] \quad (1)$$

The private parameter ϵ represents the privacy level, with larger ϵ indicating weaker privacy protection provided by $\mathcal{M}$. To satisfy the condition of LDP, $\mathcal{M}$ must effectively hide all differences between data entries, including the difference between the maximum and minimum values. In practice, this often requires adding a significant amount of noise, leading to reduced usability.

Definition 2 (Distance-based Local Differential Privacy). *An obfuscation mechanism $\mathcal{M} : \mathbb{D} \longrightarrow \mathbb{O}$ satisfies ϵ-dLDP with privacy level ϵ ($\epsilon \geq 0$) if for any $x, x' \in \mathbb{D}$ such that $|x - x'| \leq t$, and for any possible output $y \in \mathbb{O}$, we have*

$$\Pr[\mathcal{M}(x) = y] \leq e^{t\epsilon} \cdot \Pr[\mathcal{M}(x') = y] \quad (2)$$

According to this definition, the indistinguishability between any two sensitive data points decreases as the distance t between them increases [26]. This strikes a balance between privacy and utility and is more suitable for practical application scenarios.

Theorem 1 (Post-Processing [35]). *Let $\mathcal{M} : \mathbb{D} \longrightarrow \mathbb{O}$ be an ϵ-dLDP obfuscation mechanism, and $f : \mathbb{O} \longrightarrow \mathbb{O}'$ be any randomized function. Then, $f \circ \mathcal{M}$ remains ϵ-dLDP.*

Immunity to post-processing is a key property of differential privacy, meaning that arbitrary transformations can be performed on the output without compromising privacy guarantees.

2.3 Learned Indexes

Similar to PIR schemes that employ filters for keyword PIR, index structures (e.g., B+tree [21] or learned indexes) can also be used to map keywords to their corresponding locations before retrieval. Learned indexes, first introduced by Kraska et al. [51], utilize data distribution characteristics to build more efficient index structures. Specifically, it focuses on data distribution and rank to map a given key to its corresponding memory address [33, 53, 84, 86, 91, 92]. This operation can be understood as a cumulative distribution function (CDF) applied to the key distribution.

Most learned indexes follow a similar structure and search process. Given a set of key-value pairs, they use a construction algorithm to create a tree structure, typically consisting of a root node, one or more levels of internal nodes, and leaf nodes that manage the key-value pairs. Each node contains a simple linear model and metadata representing the distribution of a subset of the entire dataset. To query a key, the learned index performs a top-down traversal. At each level, a model is used to predict the next node's location to be accessed at the next level, continuing until a leaf node is reached. The model in the leaf node predicts the position (i.e., rank) of the query key, and a last-mile search is performed within a small range to determine the exact key location.

3 Motivation

We focus on a scenario where a server hosts a public key-value store, allowing users to retrieve values by their keys without revealing which specific keys they are querying. Such scenarios are common, including untraceable browsing [44, 50, 85], contact discovery [22, 32], password leakage detection [78], anonymous messaging [13, 14, 52, 62], etc. For example, at WWDC 2024, Apple introduced a live caller ID lookup feature that enables users to obtain information about incoming calls from a public dataset without revealing the queried phone number to the server. This feature helps safely block nuisance calls, offering both security and convenience.

In such scenarios, the server is typically the data owner. The users can privately perform read-only lookups on the public key-value store but are usually not allowed to modify the data. Only the server may periodically update its data as needed. This is fundamentally different from the cases where the users outsource their own databases to untrusted servers. But we emphasize that it is still a common and important problem in the real world, as exemplified above. For example, an investor may wish to privately retrieve information about a particular stock, such as its trading record, without intending to update any public data. However, these queries often involve sensitive information (e.g., investment interests), necessitating privacy guarantees.

Most existing approaches rely on PIR schemes to support such scenarios. As discussed in Section 2.1, PIR schemes usually use an encrypted one-hot bitmap to represent the location of the desired key-value pair, and perform homomorphic inner products with all key-value pairs in the dataset, thereby concealing the queried key. However, their computational cost scales with the size of the database, making them impractical for large datasets. For example, Pantheon [7] takes several hours to execute a query on a dataset containing 16,777,216 key-value pairs. Although Pantheon can parallelize computations by distributing the dataset across multiple machines, this does not fundamentally reduce the computational complexity of each query, leaving the scalability problem unresolved.

In this paper, we argue that such impractical performance of existing schemes stems from their rigid guarantees of full security, which require all key-value pairs to be involved in computation or transmission. Current PIR schemes cannot bypass this requirement, even with optimized cryptographic algorithms or preprocessing. However, this requirement is not always necessary in practical applications. In real-world scenarios, users' security requirements are independent of the dataset size and often remain fixed. For example, when protecting a user's address, the user may be comfortable revealing her country (e.g., the United States) but not her city (e.g., San Francisco). This allows the query to be executed securely over all addresses within the country, rather than over the entire global dataset. Moreover, in a geographic database like Open Street Map, which is keyed by latitude and longitude, nearby data items in the database are geographically close to each other. Practical applications often involve scenarios where the storage location in the database is correlated with the value of the key. Meanwhile, such "weaker" security is common in industrial solutions, which often use hash functions to partition the complete dataset into smaller subsets. However, such methods not only leak the bucket ID where the queried key is located but also fail to provide provable privacy guarantees in general scenarios. For example, if a bucket contains only a single entry, the queried key is immediately exposed.

To address these issues, we believe it is essential to offer flexible degrees of relaxed security based on the user's specific needs. This flexibility should still offer provable theoretical guarantees at all security levels. With a solid theoretical foundation, users can feel comfortable using more practical solutions that are not fully secure. To achieve this goal, the first requirement is to formally define how to relax security levels with theoretical guarantees. Then, a practical system design is needed to realize flexible privacy and ensure efficient query processing at all security levels. In addition, it should not impose significant computational or storage burdens on users, meaning offline computation and storage should be minimized.

4 Relaxed Security for Private Retrieval

Our objective is to develop a flexible and scalable framework that enables users to privately retrieve the value associated with a querying key from a public key-value store. The primary security goal is to hide the user's lookup queries from the server, while the server's key-value data points are public. Our scenario resembles that of the keyword PIR [7, 13, 25, 57], with the key distinction that it permits the flexibility of enforcing different levels of relaxed security based on specific user requirements. In this section, we formalize our novel definition of relaxed security and the corresponding threat model.

4.1 Problem Formulation

Let $DB = \{(k_0, v_0), (k_1, v_1), \ldots, (k_{n-1}, v_{n-1})\}$ denote the set of public key-value pairs held by the server, where the key set $K = \{k_0, k_1, \ldots, k_{n-1}\}$ serves as the primary key of the database (no duplication). DB is sorted by primary keys. Each key-value pair is stored in plaintext with a uniform length to prevent attackers from inferring the queried key based on length variations. Let k_{target} be the querying key of the user, and v_{target} be the corresponding value. We discuss our problem below in terms of correctness and privacy.

4.1.1 Correctness. If $k_{\text{target}} \in K$, the server must return the corresponding v_{target}; otherwise, it returns null, allowing the client to readily verify that k_{target} does not exist.

4.1.2 Privacy: Distance-based indistinguishability. Our privacy definition aims to protect query privacy within a distance less than t in the database DB. Here, distance refers to the difference in storage positions of two querying keys within the sorted key-value store, with t representing the maximum allowable distance specified by the client. Let q_i and q_j represent queries corresponding to keys k_i and k_j, respectively (possibly chosen by the adversary), where $|i - j| \leq t$. The user selects one query to execute, while the adversary observes the resulting events, denoted by O_i and O_j (e.g., the key-value pairs involved in the server-side computation), and attempts to distinguish which query was executed.

Inspired by LDP introduced in Section 2.2, we define a scheme as achieving distance-based indistinguishability (denoted as ϵ-dist indistinguishability) for a given maximum allowable distance t ($t > 0$), if there exists a non-negative constant ϵ such that for any pair of queries q_i and q_j where $|i - j| \leq t$ ($i, j \in [0, n)$), and for all possible adversarial observations O in the observation space Ω, the following condition holds:

$$\Pr[O \mid q_i] \leq e^{\epsilon} \cdot \Pr[O \mid q_j] \tag{3}$$

where $\Pr[O \mid q_i]$ represents the probability of the observation O given that q_i has been issued. This ensures that the adversary cannot distinguish between queries q_i and q_j based on observed events, thereby maintaining distance-based indistinguishability. The parameters ϵ and t together determine the security guarantees, where ϵ is the privacy parameter and t is the maximum allowable distance between two queries for the indistinguishability guarantee to hold. When t is fixed, increasing ϵ weakens the indistinguishability, resulting in more relaxed security guarantees. Conversely, when ϵ is fixed, a larger t strengthens the security guarantees. In Femur, the ratio of these two parameters determines the expected number of data points within the obfuscation range, as discussed in Section 7.

To minimize user burden, we set the default value of ϵ to 2^{-6}, since t is generally more intuitive for users to adjust. This empirical default value of ϵ, chosen as a power of 2, aligns with the highest security levels in prior DP-related work [54]. Users can adjust t to specify how many neighboring queries should be included to make the real query indistinguishable among them, allowing for a trade-off between privacy and performance based on individual needs. For example, suppose $DB = \{(k_0, v_0), (k_1, v_1), \ldots, (k_{99}, v_{99})\}$, and the user queries $k_i = k_{20}$ and sets t to 10. In this case, any potential output corresponding to a query q_j for any key within the range $[k_{10}, k_{30}]$ and the query q_{20} for k_{20} satisfies Equation (3). This ensures that adversaries cannot determine which specific key the user is querying within $[k_{10}, k_{30}]$, even when observing the access pattern.

Intuitively, we can obfuscate the real query by adding fake queries. Let S_i represent the set of queries that includes q_i and extra fake queries generated by the client through a perturbation mechanism that satisfies ϵ-dist indistinguishability, where $|S_i|$ is the number of queries. Then, the probability of correctly guessing q_i is $\frac{1}{|S_i|}$. If $|S_i| = n$, full security and maximum uncertainty are achieved. If $|S_i| < n$, it corresponds to relaxed quantity-based agnosticism, reducing the uncertainty. Finally, if $|S_i| = 1$, no security is provided, as the adversary can directly infer the query.

4.2 Threat Model

The user locally owns a trusted client machine. It aims to query some data in the public key-value store on an untrusted server. The stored data is in plaintext. A semi-honest adversary may control the server and want to steal sensitive information about which item is returned. The adversary can monitor incoming queries from users, observe server responses, and see any operations performed by the server, including which data point is involved in the computations (i.e., the data access patterns). It can measure the execution time, which depends on the number of queries but is not affected by which specific keys are queried. It can also monitor, record, and analyze data communication between the client and the server, even utilizing any prior knowledge and observations, as well as the query history. However, the adversary cannot break cryptographic schemes like AES. This ensures that the adversary cannot decrypt the encrypted data (i.e., the querying key and the returned result). Besides, it will not tamper with any data or code executed on the server.

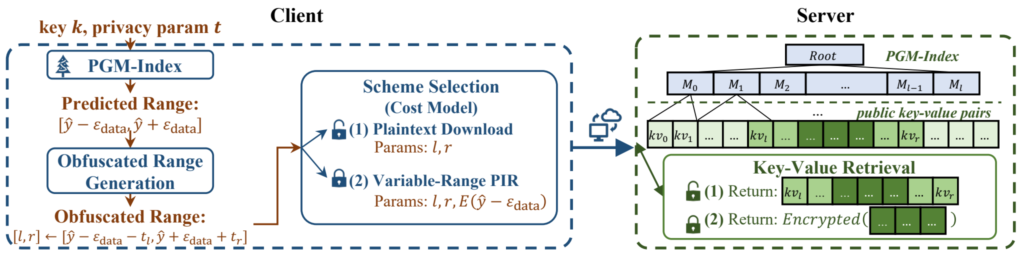

Client
Server
key k, privacy param t
PGM-Index
Predicted Range:
$[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$
Obfuscated Range Generation
Obfuscated Range:
$[l, r] \leftarrow [\hat{y} - \varepsilon_{\text{data}} - t_l, \hat{y} + \varepsilon_{\text{data}} + t_r]$
Scheme Selection (Cost Model)
(1) Plaintext Download
Params: l, r
(2) Variable-Range PIR
Params: $l, r, E(\hat{y} - \varepsilon_{\text{data}})$
Root
PGM-Index
M_0 M_1 M_2 ... M_{l-1} M_l
public key-value pairs
kv_0 kv_1 kv_l kv_r
Key-Value Retrieval
(1) Return: kv_l ... kv_r
(2) Return: Encrypted(...)

Fig. 1. The Core Components of Femur Framework.

5 Design Overview

Femur allows users to flexibly control the privacy-preserving granularity of queries for efficiently processing. Femur has two phases: offline initialization and online query phase. The initialization phase is executed offline only once when there is no database update, allowing users to gather essential information from the server. The online query phase includes the entire process from client-side query generation to receiving results from the server. In this section, we outline the key components and workflow of Femur.

5.1 Offline Initialization Phase

When a client arrives, it begins by exchanging one-time information with the server during the initialization phase. Specifically, the server provides the client with essential details about the database, including the number of key-value pairs, the bit length of each pair, and the available retrieval schemes with their corresponding parameters. Besides, the server supplies an auxiliary index to allow the client to locally convert querying keys into (approximate) server storage locations. Femur offers two optional retrieval schemes: *plaintext download* with no extra parameters, and *variable-range PIR*, which requires three cryptographic parameters: N, p, and q (as detailed in Section 8.2). The client then constructs the public and private cryptographic keys needed to encrypt/decrypt queries in the variable-range PIR scheme. Upon completing this step, the client sends its cryptographic public key to the server. The server retains each client's public key for future interactions.

The auxiliary index structure provided by the server is the PGM-index, a space-efficient learned index that does not store the actual key values but only the model parameters (e.g., slopes, intercepts, and partitioning keys). This reduces the data transmitted to clients to tens of KB to a few MB, greatly cutting initialization overhead.

If the key-value store is updated (usually periodically by the server; Section 3), the initialization phase needs to be re-executed. Particularly, the new auxiliary index structure needs to be synchronized with the clients. Section 9 further discusses the details.

5.2 Online Query Phase

The online phase includes the entire query lookup process. After determining the querying key k and the desired privacy level t, the client generates a secure query range utilizing three components provided by Femur: key-to-position conversion, obfuscated range generation, and scheme selection. The obfuscated range boundaries and the selected retrieval scheme are then sent to the server. The server processes the query using the specified scheme and returns the corresponding key-value pairs. Finally, the client retrieves the desired value by performing a simple verification. Below we briefly describe the main building blocks of Femur, as shown in Figure 1.

5.2.1 Key-to-Position Conversion. We employ the PGM-index, a learned index, to map the user's querying key to its possible locations within the dataset (Section 6). It guarantees that the key will be found within these positions unless it does not exist in the database. The parameter $\varepsilon_{\text{data}}$ in the PGM-index determines the range of possible positions. For a given key k, the PGM-index outputs a predicted position $\hat{y}$, and the possible locations are then bounded by $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$.

5.2.2 Obfuscated Range Generation. In this step, we efficiently convert the range $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$ into an obfuscated range $[l, r] = [\hat{y} - \varepsilon_{\text{data}} - t_l, \hat{y} + \varepsilon_{\text{data}} + t_r]$. Specifically, this obfuscated range is generated using a noise generation mechanism that satisfies distance-based indistinguishability (Section 7). It ensures that the server cannot infer the real key being queried, while eliminating the need for all data points to be involved in the computation.

5.2.3 Scheme Selection. Once the obfuscated range is determined, our framework uses a cost model to select the most efficient retrieval scheme for the current query between plaintext downloads and variable-range PIRs (Section 8). The cost model is lightweight and can be executed efficiently on the client side, leading to minimal overhead. For plaintext downloads, only the unencrypted boundaries l and r are sent to the server. For variable-range PIRs, besides l and r, the left boundary $(\hat{y} - \varepsilon_{\text{data}})$ of the predicted range is encrypted and also sent to the server.

5.2.4 Server-Side Query Processing. Upon receiving the query request, the server processes the query using the designated retrieval scheme (Section 8). For plaintext downloads, the server sends the key-value pairs in the range $[l, r]$ directly back to the client in unencrypted form. For variable-range PIR, the server performs homomorphic encryption computations and returns a single ciphertext containing all key-value pairs in $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$. We carefully control $\varepsilon_{\text{data}}$ and the plaintext encoding scheme to ensure that the queried pair is always returned. Note that the server is unaware of the specific granularity of the querying key or the user's privacy settings throughout the retrieval and transmission process.

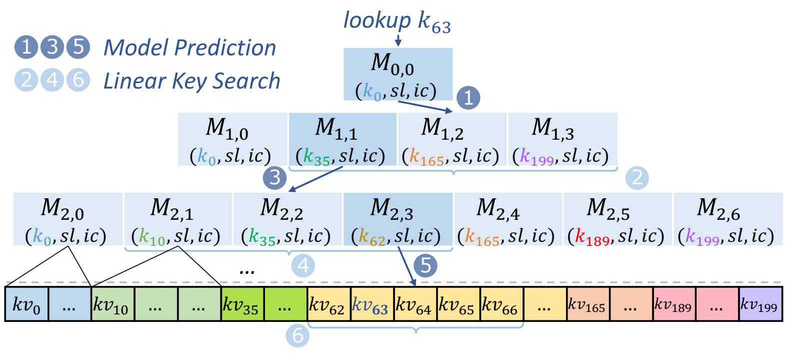
6 Key-to-Position Conversion

To facilitate privacy-preserving data retrieval on the server side, the querying key needs to be converted into the corresponding position of the key-value pair in the dataset, as in traditional keyword PIR schemes. Database index structures [15, 21, 33, 40, 53, 84, 89–92] are exactly designed for such key-to-position conversion. In Femur, we propose to apply PGM-indexes to perform this conversion on the client side, allowing direct processing of the querying key in plaintext. We will discuss the advantages and disadvantages of storing index structures on the client side in Section 6.1, along with the specific requirements for selecting a suitable index structure for our scenario. Then, we introduce the details of our choice, the PGM-index, in Section 6.2.

6.1 Client-Side Indexes

In Femur, while the index structure used for key-to-position conversion resides on the client side, it is created and maintained by the server, with the client only storing a static copy. This static copy requires no maintenance on the client side and functions like pre-downloaded hints in other PIR schemes [25, 31, 45]. Sharing this index of the public database poses no security concerns.

The benefits of this approach align with those of PIR schemes, as static index copies can be downloaded during the initialization phase and serve as a local cache to accelerate query execution during the online phase. Specifically, in Femur, the client-side index efficiently maps query keys to predicted locations, which are then expanded using our obfuscated range generation method. This narrows the range of key-value pairs involved in subsequent computations, significantly enhancing



❶❸❺ Model Prediction
❷❹❻ Linear Key Search
lookup k_{63}
$M_{0,0}$ (k_0, sl, ic)
$M_{1,0}$ (k_0, sl, ic)
$M_{1,1}$ (k_{35}, sl, ic)
$M_{1,2}$ (k_{165}, sl, ic)
$M_{1,3}$ (k_{199}, sl, ic)
$M_{2,0}$ (k_0, sl, ic)
$M_{2,1}$ (k_{10}, sl, ic)
$M_{2,2}$ (k_{35}, sl, ic)
$M_{2,3}$ (k_{62}, sl, ic)
$M_{2,4}$ (k_{165}, sl, ic)
$M_{2,5}$ (k_{189}, sl, ic)
$M_{2,6}$ (k_{199}, sl, ic)
kv_0 ... kv_{10} ... ... kv_{35} ... kv_{62} kv_{63} kv_{64} kv_{65} kv_{66} ... kv_{165} ... kv_{189} ... kv_{199}

Fig. 2. An Example of the PGM-index.

query performance. In contrast, prior schemes speed up computations by pre-computing parts of the lookup process but fail to reduce the overall data volume processed.

The main drawback of storing a static copy on the client side is the potential for index staleness due to database updates. Similar to PIR solutions [66], Femur restricts to periodic batch updates for keys (Section 9). Thus the overhead for the client to fetch the latest index from the server is insignificant. Moreover, selecting a compact index could mitigate this issue by reducing the time required to download updated versions, further improving efficiency.

Consequently, we have two requirements for the index structure. First, to map querying keys to database locations, the index should provide item-level granularity rather than block-level granularity. This allows each data point to be obfuscated at the finest granularity, thus reducing the noise (i.e., the number of fake queries) needed for indistinguishability. Traditional structures like B+trees [21], which organize data into pages or large leaf nodes, map querying keys to page IDs. Even if key-value pairs are evenly distributed, noise must be introduced at the page level to satisfy privacy requirements, forcing the obfuscated range to include entire pages and increasing the volume of accessed data. Second, the size of the index must remain small to reduce the overhead of the client when downloading the index during initialization and after database updates. Large structures (e.g., hints in the offline phase of many PIR schemes [25, 31, 45, 60, 95]) can become bottlenecks, especially in scenarios where multiple clients simultaneously access the server, leading to bandwidth constraints.

We find that learned indexes can effectively fulfill these two requirements. First, learned indexes directly map a querying key to a predicted position in the entire sorted array, providing item-level prediction granularity. We will thoroughly discuss the theoretical differences between adding noise to these two indexes in Section 7. Second, they consume significantly less memory compared to B+trees, achieving compression rates for non-leaf nodes up to 2,000× smaller than the internal nodes of a B+tree [58, 84, 92].

6.2 PGM-Indexes

We integrate a state-of-the-art space-efficient learned index, the PGM-index [40], into Femur to facilitate key-to-position conversion with item-level granularity flexibility. As in Figure 2, the PGM-index consists of multiple layers of simple linear regression models, where each model node is defined by only two parameters: slope and intercept (i.e., $\hat{y} = \text{slope} \times k + \text{intercept}$), as well as the minimum key (partitioning key) of the sub-dataset it manages. The index has two hyperparameters: $\varepsilon_{\text{model}}$ and $\varepsilon_{\text{data}}$, which determine the maximum allowable prediction errors for all non-leaf nodes and all leaf nodes, respectively. For a predicted position $\hat{y}$, the desired key-value pair is guaranteed to reside in the range $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$, otherwise this pair is not in the key-value store.

We provide an example of the PGM-index with $\varepsilon_{\text{model}} = 1$ and $\varepsilon_{\text{data}} = 2$ in Figure 2. To look up a given key k_{63}, model $M_{0,0}$ uses its slope and intercept to predict the next layer's position: $\hat{y} = sl \cdot k_{63} + ic = 2$. The next-layer models within the range $[\hat{y} - \varepsilon_{\text{model}}, \hat{y} + \varepsilon_{\text{model}}] = [M_{1,1}, M_{1,3}]$ are searched, locating $M_{1,1}$ (since $k_{35} \le k_{63} \le k_{165}$). Then, $M_{1,1}$ predicts $[M_{2,1}, M_{2,3}]$ for layer 2 and $M_{2,3}$ is identified similarly. Recursively, $M_{2,3}$ predicts position 64, narrowing the last-mile search range to $[kv_{62}, kv_{66}]$ with $\pm\varepsilon_{\text{data}}$. Finally, kv_{63} is successfully located using binary search.

For each fixed dataset, the PGM-index is constructed using a bottom-up hierarchical approach. The bottom layer scans the ordered key-value pairs with a greedy algorithm of time complexity $O(n)$. As described in [40], constructing the linear models reduces to constructing the convex hull of a set of points. Starting at index $i = 0$, the algorithm checks whether the current i-th key-value pair can be added to the current latest j-th model without exceeding $\varepsilon_{\text{data}}$. If true, it moves to the next key-value pair. Otherwise, a linear model (i.e., slope and intercept) is determined by the line that splits the rectangle into two equal-sized halves. Then, a new model $j + 1$ is initialized starting from index i. This ensures that each model manages key-value pairs constrained by a rectangle of height $2\varepsilon_{\text{data}}$. This process continues until all key-value pairs are scanned. Upper layers follow a similar approach but operate on sub-datasets, where prediction errors are constrained by $\varepsilon_{\text{model}}$. For example, when constructing layer 0, the algorithm processes the pairs $\{\{k_0, 0\}, \{k_{35}, 1\}, \{k_{165}, 2\}, \{k_{199}, 3\}\}$. Construction continues until a single model remains at the top layer. More details are provided in [40]. To maintain simplicity within Femur, we use the default values of the PGM-index ($\varepsilon_{\text{data}} = 64$, $\varepsilon_{\text{model}} = 4$) across all datasets, avoiding parameter tuning complexity.

The PGM-index is compact, reducing communication overhead during initialization. During the online phase, it can quickly convert a queried key to a predicted range containing the desired key's location. The size of this range is $2 \times \varepsilon_{\text{data}} + 1$, where all positions except the correct one act as fake queries. However, the predicted range does not yet satisfy distance-based indistinguishability, which will be addressed in the next section.

7 Obfuscated Range Generation

In this section, we describe how to generate obfuscated ranges that satisfy distance-based indistinguishability. Femur allows users to flexibly specify a relaxed security level for each query. It then employs a noise-generation mechanism, which expands the predicted range $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$, derived from the key-to-position conversion step, to a wider obfuscated range $[l, r]$. To achieve both high performance and sufficient security, the length of the obfuscated range is set to the minimum value that still ensures distance-based indistinguishability depending on the user-specified security level. Specifically, any two queries whose keys are within a specified distance t are indistinguishable to the adversary. Larger t requires a wider obfuscated range. Then, l and r are sent to the server, and only data points within this range need to be involved in the computation and communication.

We first employ the exponential mechanism, a widely used approach in differential privacy, to generate noise. The exponential mechanism assigns probabilities to points in a given set $\mathbb{D}$, typically based on the distance between the point $i \in \mathbb{D}$ and the real value x:

$$p_{x,i} = \Pr[o = i] = \frac{e^{-|x-i| \cdot \epsilon_{\text{dp}}/4t}}{\sum_{j \in \mathbb{D}} e^{-|x-j| \cdot \epsilon_{\text{dp}}/4t}} \tag{4}$$

Points closer to the real value would have higher probabilities of being chosen, thereby maintaining indistinguishability while reducing the impact on performance. A point is then randomly sampled as the new boundary based on these probabilities. This process is carried out independently for the left and right boundaries, with $[0, \hat{y} - \varepsilon_{\text{data}}]$ and $[\hat{y} + \varepsilon_{\text{data}}, n)$ as the given $\mathbb{D}$, respectively, and produces the obfuscated range $[l, r]$. We then have the following theorem for the privacy guarantee.

Theorem 2. *The exponential mechanism described above provides ϵ_{dp}-dist indistinguishability privacy guarantee for any pair of values $x, x' \in \mathbb{D}$, where $|x - x'| \leq t$, and, $t, \epsilon_{dp} > 0$.*

Proof. In our scenario, the observations O visible to the adversary consist of the boundaries (i.e., the outputs l and r). We show that our algorithm provides a $\frac{\epsilon_{\text{dp}}}{2}$-dist indistinguishability privacy guarantee for each boundary. Let o be one of the two boundaries l or r, and $\mathbb{D}$ be the set of potential new values that it is obfuscated to (i.e., $[0, \hat{y} - \varepsilon_{\text{data}}]$ or $[\hat{y} + \varepsilon_{\text{data}}, n)$).

$$\frac{\Pr[O = o|x]}{\Pr[O = o|x']} = e^{\frac{\epsilon_{\text{dp}}}{4t}(|x'-o|-|x-o|)} \cdot \frac{\sum_{j\in\mathbb{D}} e^{-|x'-j|\cdot\epsilon_{\text{dp}}/4t}}{\sum_{j\in\mathbb{D}} e^{-|x-j|\cdot\epsilon_{\text{dp}}/4t}} \tag{5}$$

By applying the triangle inequality $|x' - o| = |x' - x + x - o| \leq |x' - x| + |x - o|$, we can get:

$$e^{\frac{\epsilon_{\text{dp}}}{4t}(|x'-o|-|x-o|)} \leq e^{\frac{\epsilon_{\text{dp}}}{4t}|x'-x|} \leq e^{\frac{\epsilon_{\text{dp}}\cdot t}{4t}} \tag{6}$$

Similarly, $-|x' - j| \leq |x' - x| - |x - j|$, and we get:

$$\begin{aligned} \sum_{j\in\mathbb{D}} e^{-|x'-j|\cdot\epsilon_{\text{dp}}/4t} &\leq \sum_{j\in\mathbb{D}} \left(e^{|x'-x|\cdot\epsilon_{\text{dp}}/4t} \cdot e^{-|x-j|\cdot\epsilon_{\text{dp}}/4t}\right) \\ &\leq e^{|x'-x|\cdot\epsilon_{\text{dp}}/4t} \cdot \sum_{j\in\mathbb{D}} e^{-|x-j|\cdot\epsilon_{\text{dp}}/4t} \\ &\leq e^{\frac{\epsilon_{\text{dp}}\cdot t}{4t}} \cdot \sum_{j\in\mathbb{D}} e^{-|x-j|\cdot\epsilon_{\text{dp}}/4t} \end{aligned} \tag{7}$$

Combining both terms, we have the overall bound:

$$\frac{\Pr[O = o|x]}{\Pr[O = o|x']} \leq e^{\frac{\epsilon_{\text{dp}}\cdot t}{4t}} \cdot e^{\frac{\epsilon_{\text{dp}}\cdot t}{4t}} \leq e^{\frac{\epsilon_{\text{dp}}}{2}} \tag{8}$$

Since the exponential mechanism is used for both boundaries, the overall algorithm satisfies the ϵ_{dp}-dist indistinguishability. □

The ratio of the distance t and the privacy parameter ϵ_{dp} controls the size of the obfuscated range. For smaller $\frac{t}{\epsilon_{\text{dp}}}$, the obfuscated range is probabilistically reduced, which provides more relaxed privacy but saves the computational and communication costs.

It is worth noting that the exponential mechanism involves calculating probabilities for all points in $\mathbb{D}$ and then sampling from them, which is costly. To address this issue, we simulate the exponential mechanism by employing the widely used discrete Laplace mechanism [54], as defined in Definition 3, with $\lambda = 2t/\epsilon_{\text{dp}}$. The process for obfuscated range generation using the discrete Laplace mechanism is outlined in Algorithm 1.

Definition 3 (Discrete Laplace Distribution). *The discrete Laplace distribution with a scale parameter λ is denoted as $\text{Lap}_{\mathbb{Z}}(\lambda)$, where $\mathbb{Z}$ represent the set of integers. If a random variable X follows $Lap_{\mathbb{Z}}(\lambda)$, its probability distribution is defined as:*

$$\forall x \in \mathbb{Z}, \quad \Pr[X = x] = \frac{e^{1/\lambda} - 1}{e^{1/\lambda} + 1} \cdot e^{-|x|/\lambda} \tag{9}$$

We then prove that adding noise according to a discrete Laplace distribution satisfies distance-based indistinguishability.

Algorithm 1 $\text{Lap}_{\mathbb{Z}}$-based Obfuscated Range Generation

```
Input: $\hat{y}$, $\varepsilon_{\text{data}}$, dataset size $n$, privacy parameter $\epsilon_{\text{dp}} > 0$, $t > 0$
Output: obfuscated range $[l, r]$
 1: function ModularLap$_{\mathbb{Z}}$($\mathbb{D}, \epsilon_{\text{dp}}, t$)
 2:     $x \leftarrow Lap_{\mathbb{Z}}(\frac{2t}{\epsilon_{\text{dp}}})$
 3:     $x \leftarrow x\% |\mathbb{D}|$
 4:     return $\mathbb{D}[x]$
 5: end function
 6: $\mathbb{D}_l \leftarrow [\hat{y} - \varepsilon_{\text{data}}, \cdots, 1, 0, n-1, n-2, \cdots, \hat{y} + \varepsilon_{\text{data}}]$
 7: $\mathbb{D}_r \leftarrow [\hat{y} + \varepsilon_{\text{data}}, \cdots, n-2, n-1, 0, 1, \cdots, \hat{y} - \varepsilon_{\text{data}}]$
 8: $l \leftarrow \text{ModularLap}_{\mathbb{Z}}(\mathbb{D}_l, \epsilon_{\text{dp}}, t)$
 9: $r \leftarrow \text{ModularLap}_{\mathbb{Z}}(\mathbb{D}_r, \epsilon_{\text{dp}}, t)$
10: if $l \le \hat{y} - \varepsilon_{\text{data}} \le \hat{y} + \varepsilon_{\text{data}} \le r$ then
11:     return $[l, r]$
12: else if $r < l \le \hat{y} - \varepsilon_{\text{data}}$ or $\hat{y} + \varepsilon_{\text{data}} \le r < l$ then
13:     return $[l, n-1] \cup [0, r]$
14: else
15:     return $[0, n-1]$
16: end if
```

Theorem 3. ***Lap$_{\mathbb{Z}}$-based Obfuscated Range Generation*** *provides ϵ_{dp}-dist indistinguishability privacy guarantee for any pair of values x_1, x_2, where $|x_1 - x_2| \le t$, and $t, \epsilon_{dp} > 0$.*

Proof. This algorithm provides a $\frac{\epsilon_{\text{dp}}}{2}$-dist indistinguishability privacy guarantee for each boundary (i.e., $\hat{y} - \varepsilon_{\text{data}}$, $\hat{y} + \varepsilon_{\text{data}}$). Let l_1 and l_2 be the left boundaries of x_1 and x_2. The probability ratio of l_1 and l_2 being randomized to the same output value o is:

$$\frac{\Pr[l_1 + N_1]}{\Pr[l_2 + N_2]} = \frac{\Pr[N_1 = o - l_1]}{\Pr[N_2 = o - l_2]} = \frac{e^{-|o-l_1|/\lambda}}{e^{-|o-l_2|/\lambda}} \le e^{\frac{\epsilon_{\text{dp}}}{2}} \tag{10}$$

where N_1 and N_2 follow $\text{Lap}_{\mathbb{Z}}(\lambda)$. Since this mechanism is used for both boundaries, the overall algorithm satisfies the ϵ_{dp}-dist indistinguishability[1]: $\epsilon_{\text{overall}} = \epsilon_l + \epsilon_r = \epsilon_{\text{dp}}/2 + \epsilon_{\text{dp}}/2 = \epsilon_{\text{dp}}$. □

Theorem 3 guarantees that for any two queries, q_i and q_j ($|i - j| \le t$), the probability of distinguishing between them is bounded by $e^{\epsilon_{\text{dp}}}$. This covers two cases: when $q_i \neq q_j$ and when the same query is submitted repeatedly (i.e., $q_i = q_j$). In both cases, the adversary will be unable to distinguish the queries.

Since the negative numbers sampled from $\text{Lap}_{\mathbb{Z}}(\lambda)$ are modulo positive, the expected amount of noise added to one boundary is $\mathbb{E}[X \mid X > 0] = \frac{1}{1-e^{-1/\lambda}} \approx 2t/\epsilon_{\text{dp}}$. Accounting for both boundaries, the expected length of the obfuscated range becomes $4t/\epsilon_{\text{dp}} + 2\varepsilon_{\text{data}} + 1$, where the last two terms specify the output range from the PGM-index. This length is capped at the total number of key-value pairs, i.e., n. The total number of fake queries equals this length minus one.

Discussion. Below, we analyze why B+tree requires more noise compared to PGM-indexes. The B+tree index only needs to transfer non-leaf nodes to the client, who uses it to obtain the ID of the leaf node (i.e., page ID) containing the querying key, as well as the IDs of the key-value pairs at the left and right boundaries of this page ($l_{\text{B+tree}}$ and $r_{\text{B+tree}}$). Note that noise must be added at the

[1]Note that our proof now demonstrates that distance-based indistinguishability holds for two boundaries derived from the predicted positions ($\hat{y}$). For practical use, the user should set $t' = t - 2\varepsilon_{\text{data}}$ as the final distance. Since $|\hat{y}_i - \hat{y}_j| \le |i - j| + 2\varepsilon_{\text{data}} \le t + 2\varepsilon_{\text{data}}$, this guarantees that the proposed mechanism preserves distance-based indistinguishability when applied to queries.

page-level granularity. Otherwise, the adversary may infer that the querying key is not located in any incomplete page within the range (e.g., the first or last page), leaking information.

Assume m is the number of key-value pairs per page, and key-value pairs are evenly distributed within each page. Thus page ID $= \frac{\text{item ID}}{m}$. The distance in the B+tree can be defined as $t' = \lceil \frac{t}{m} \rceil$. The boundaries $l_{\text{B+tree}}$ and $r_{\text{B+tree}}$ are then extended using the same noise mechanism ($\text{Lap}_{\mathbb{Z}}(\lambda')$), where $\lambda' = 2t'/\epsilon_{\text{dp}}$. Consequently, the expected length of the obfuscated range becomes $\frac{4m}{\epsilon_{\text{dp}}} \cdot \lceil \frac{t}{m} \rceil + m$.

Compared to the PGM-index, the B+tree generally requires more noise. For example, the leaf nodes of a B+tree typically occupy 4 KB, containing 256 key-value pairs of 16 bytes each. When using default parameters ($\varepsilon_{\text{data}} = 64$, $\epsilon_{\text{dp}} = 2^{-6}$) and a user-selected $t = 100$, the expected obfuscation range length for a B+tree is 65,792, while for a PGM-index, it is 25,729. When $t = 10,000$, the expected lengths increase to 2,621,696 for a B+tree and 2,560,129 for a PGM-index, respectively, indicating that the PGM-index requires less noise overall, especially for fine-grained security levels. More empirical results are provided in Section 10.3.4.

8 Key-Value Retrieval

In this section, we describe our server-side adaptive key-value retrieval module, including two schemes: plaintext download and variable-range PIR. Additionally, we present a lightweight cost model to help make fast and efficient decisions between these two schemes based on query characteristics and system configurations.

8.1 Plaintext Download

Plaintext download is a straightforward method for retrieving key-value pairs, where the server directly sends the key-value pairs within the obfuscated range to the client, which performs the searches locally. The client only needs to provide the boundaries l and r. This approach satisfies the security requirements, as the server only sees the boundaries that have been obfuscated to ensure distance-based indistinguishability, and remains unaware of the exact key the client intends to retrieve.

The main advantage of plaintext download is server-side efficiency, as the server can immediately transmit the specified key-value pairs over the network without additional computation. However, this approach is bandwidth-intensive, which can lead to performance bottlenecks in low-bandwidth or high-traffic environments. Besides, plaintext download can further reduce the data to be transferred through compression techniques [5, 55, 70, 88]. Although we do not apply compression in our experiments, it can be integrated as an optional optimization.

8.2 Variable-range PIR

To handle low-bandwidth scenarios, we propose a novel variable-range PIR scheme. This enhanced scheme allows performing "tiny PIR" on a specific range of each query without re-preprocessing. Our design significantly improves server-side computational efficiency when querying large datasets with relaxed privacy levels.

8.2.1 Underlying Cryptography. We leverage the SEAL FHE library, which is based on the BFV FHE cryptosystem, to perform "tiny PIR". This method encodes several data points into a single plaintext, enabling vectorized homomorphic operations. Furthermore, each retrieval returns the key-value pairs packed within the same plaintext, facilitating simultaneous retrieval of multiple key-value pairs. The number of key-value pairs (denoted as M) that a single plaintext can hold is determined by: $M = \frac{N}{\lceil kv_{\text{bits}}/\log_2 p \rceil}$, where the polynomial degree N and the plaintext modulus p are both internal parameters of the FHE scheme. The modulus q, which governs the noise capacity

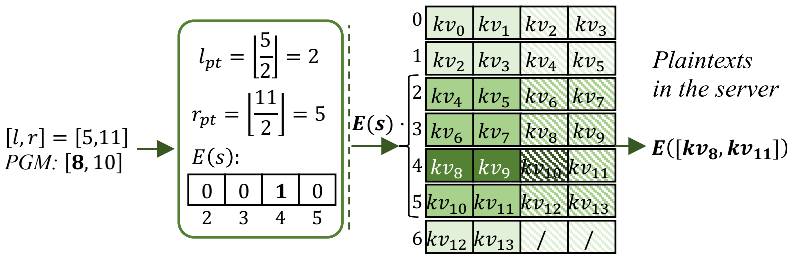

$l_{pt} = \lfloor \frac{5}{2} \rfloor = 2$
$r_{pt} = \lfloor \frac{11}{2} \rfloor = 5$
$E(s)$:
0 0 1 0
2 3 4 5
$[l, r] = [5,11]$
PGM: [8, 10]
$E(s)\cdot$
kv_0 kv_1 kv_2 kv_3
kv_2 kv_3 kv_4 kv_5
kv_4 kv_5 kv_6 kv_7
kv_6 kv_7 kv_8 kv_9
kv_8 kv_9 kv_{10} kv_{11}
kv_{10} kv_{11} kv_{12} kv_{13}
kv_{12} kv_{13} / /
Plaintexts in the server
$E([kv_8, kv_{11}])$

Fig. 3. An Example of Data Encoding and Query Processing in Variable-Range PIR.

and the overall security, can be derived from N and p. We use the recommended default values of $N = 4096$ and $\log_2 p = 20$. More details can be found in the original paper [13, 27, 38].

8.2.2 PGM-Oriented Misaligned Encoding. One specific challenge in our framework is that we must return the key-value pairs in the range $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$ as predicted by the PGM-index, rather than a single record as in classical PIR schemes. Without a careful design, the key-value pairs in this range may cross two or multiple plaintexts, all of which need to be transmitted to the client. For example, if $M = 4$ and we need to fetch $[kv_6, kv_9]$, two plaintexts corresponding to $[kv_4, kv_7]$ and $[kv_8, kv_{11}]$ must be returned.

We introduce a PGM-oriented misaligned encoding scheme. As illustrated in Figure 3, only $M/2$ key-value pairs are encoded into each plaintext, with the second half containing a duplicate of the first $M/2$ pairs from the next plaintext. This ensures that, as long as the PGM-index satisfies $2\varepsilon_{\text{data}} + 1 < \frac{M}{2} + 2$, all required key-value pairs can be retrieved in a single plaintext. This condition is easily satisfied with typical configurations, e.g., usually $\varepsilon_{\text{data}} = 64$ and $M = \frac{4096}{\lceil 128/20 \rceil} = 585$. Furthermore, this misaligned encoding can be generalized to support longer predicted ranges by increasing the overlap between adjacent plaintexts. With m key-value pairs overlapped, the condition would become $2\varepsilon_{\text{data}} + 1 < m + 2$.

8.2.3 Query Processing. Recall that $[\hat{y} - \varepsilon_{\text{data}}, \hat{y} + \varepsilon_{\text{data}}]$ is enlarged to $[l, r]$ to ensure distance-based indistinguishability. While only the key-value pairs in the first (smaller) range are returned to the user, all the pairs in the second (larger) range must be uniformly accessed and processed to obscure sensitive access patterns. The size of this larger range can vary significantly with different user-specified security requirements. This requires that the underlying PIR scheme should encode plaintexts independently (e.g., SealPIR [13] and OnionPIR [64]) without pre-computation involving the client. In this paper, we use SealPIR. But Femur can incorporate more advanced PIR interfaces in the future.

Figure 3 illustrates the steps in processing a lookup query with our variable-range PIR based on the misaligned encoding. Both the predicted and obfuscated ranges are first converted to the plaintext IDs at the client side using the encoding information (e.g., M) obtained during initialization. Here with $M = 4$, $l = 5$, and $r = 11$, the plaintext IDs are $l_{\text{pt}} = 2$ and $r_{\text{pt}} = 5$, involving four plaintexts in the computation. The predicted range is encoded into a one-hot vector s of length $(r_{\text{pt}} - l_{\text{pt}} + 1)$, where 1 indicates the target plaintext and 0 represents the others. In this case, the third element of s is set to 1, corresponding to the 4th plaintext. The client sends l_{pt}, r_{pt}, and the homomorphic encryption of s, $E(s)$, to the server. Note that the server performs homomorphic operations only on the obfuscated range. Specifically, a homomorphic inner product is performed between $E(s)$ and the plaintexts in $[l_{\text{pt}}, r_{\text{pt}}]$, producing an encrypted output containing the retrieved key-value pairs (i.e., $[kv_8, kv_{11}]$), which is then returned to the client.

Our variable-range PIR enables relaxed lookup queries with flexible security requirements without requiring re-encoding, improving utility and scalability. Additionally, misaligned encoding ensures efficient retrieval of key-value pairs in a single PIR call, reducing communication overheads. However, compared to plaintext downloads, it increases server-side computation due to the homomorphic encryption operations required for PIR requests.

8.3 Scheme Selection

We employ a cost model to efficiently choose between the two schemes. The model calculates the latency for each scheme using the following formula, as latency is the primary concern for clients:

$$C = \frac{Comm}{\text{bandwidth}} + C_{\text{compute}} \tag{11}$$

where $Comm$ represents the total communication amount of the scheme, and C_{compute} denotes the server-side computation time. This cost model is broadly applicable to many retrieval schemes.

For plaintext downloads, the communication cost is $w \times kv_{\text{bits}}$, where w is the length of the obfuscated range, and the computation time is negligible. In contrast, for variable-range PIR, the communication cost includes the encrypted one-hot vector and the ciphertext returned by the server. Its computation time is $w \times C_{\text{FHE}}$, where C_{FHE} is the time needed to perform a homomorphic computation operation on a single plaintext, which increases with the number of plaintexts involved.

Both bandwidth and C_{FHE} can be pre-determined, facilitating quick decisions during the online phase. By leveraging this cost model, we can efficiently combine different retrieval schemes, enabling faster privacy-preserving retrieval for users.

9 Supporting Updates

As discussed in Section 3, in our scenarios, database updates are performed by the server, not by the clients. This setting aligns with existing PIR schemes [25, 31, 45, 66], in which any modification necessitates re-initializing the entire PIR scheme, including re-running the offline phase. In Femur, we support two strategies to handle updates, i.e., real-time updates for *values* and periodic batch updates for *keys*. We also leverage multi-version control to synchronize server updates with clients.

For value updates, the plaintext of the updated key-value pair is retrieved from the key-value store, modified in real-time, and then stored back. Since the PGM-index is constructed based on keys only, it remains unchanged during value updates. Throughout this process, client lookup queries continue to operate as usual. If the obfuscated range of a lookup query overlaps with the updated plaintext, read-write locks are employed to ensure consistency.

For key updates, we employ periodic batch updates via a copy-on-write model. In this approach, the server initiates a new cloud instance or background thread to merge the existing key-value store with new key-value pairs, performing necessary inserts, deletes, and re-encoding operations. Only key-value pairs that need to be moved to maintain order are re-encoded. For example, if an insertion occurs at position 7 in a dataset of 10 key-value pairs, the plaintexts of the first 6 pairs remain unchanged and do not require re-encoding. Once the merge is complete, the server constructs a new PGM-index for the updated dataset. This process is done asynchronously (or offline) and does not disturb services on older versions. After the update, the server broadcasts the new PGM-index and version information to all connected clients, or clients can request it on demand. If a client queries using the old version, the server returns results from that version while notifying the client of the version inconsistency and providing the latest PGM-index. The client can then either use the old version's results or re-execute the query using the new index. In our experiments, clients re-execute queries with the latest version. Once all clients have confirmed that they have switched to the new version, the old key-value store is deleted.

10 Evaluation

In this section, we conduct a comprehensive evaluation of Femur using experiments on multiple datasets. Section 10.1 outlines the experimental setup, while Section 10.2 provides an in-depth analysis of end-to-end performance. Section 10.3 presents the explorations of the impact of various factors. Finally, Section 10.4 evaluates a realistic scenario involving Redis and update operations.

10.1 Experimental Setup

We conduct experiments on a machine with two Intel® Xeon® Platinum 8474C CPUs (2.10 GHz, 48 cores per socket) and 512 GB of RAM. All end-to-end evaluations in Section 10.2 are performed using 8 threads, and the remaining experiments use a single thread. We select the commonly used WAN setting (50 Mbps bandwidth with 30 ms roundtrip latency) as the default configuration [48, 49, 63]. We provide two additional bandwidth setups in Section 10.3.2, 100 Mbps and 10 Mbps, all with a roundtrip latency of 30 ms.

Datasets and Workloads. We use the real-world Open Street Map Coordinates (OSMC) dataset from the SOSD benchmark [58]. It contains 200 million key-value pairs representing real geographic positions. In contrast, existing PIRs [7, 25, 66] often use randomly generated datasets with at most a few million data points and thus fail to fully capture the complexity of large-scale, real-world applications. Both keys and values are 8-byte unsigned integers. This uniform length, consistent with PIR schemes, prevents the server from inferring querying keys based on lengths. We use the 200M key-value pairs in Section 10.2. We randomly select 100 keys (similar to [25]) from the dataset for client lookups during the online phase. We use a subset of the datasets in other experiments.

Metrics. Our evaluation includes four main metrics: offline/online communication volume and offline/online execution time. Communication volume refers to the total amount of data transferred between the server and the client at each phase, measured in megabytes (MB). The execution time includes both the client/server computation time and the network data transmission time, providing an end-to-end performance measure.

Femur and Baselines. We implement Femur and two keyword-PIR baselines, Chalamet [25] and Pantheon [7], in Java. Since the original Rust implementation of Chalamet does not include network communication, it cannot be evaluated directly under our client-server experiment setting. To enable a fair end-to-end evaluation, we reimplemented Chalamet in Java[2] based on its open-source implementation [2]. Both Chalamet and Pantheon are fully secure schemes, while Femur can support different security granularities. As discussed in Section 4.1.2, we keep $\epsilon_{\text{dp}} = 2^{-6}$, and specify three representative security levels as high ($t = 1,000,000$), medium ($t = 10,000$), and low ($t = 100$). We build the PGM-indexes using the complete datasets of the respective size (e.g., 200 million key-value pairs in the end-to-end evaluation) with parameters set to default values ($\varepsilon_{\text{model}} = 4, \varepsilon_{\text{data}} = 64$). To support variable-range PIR in Femur, we employ Java Native Interface (JNI) technology to call the Microsoft SEAL library (v4.1) [73] and use default settings for the BFV scheme. Our source code is publicly available [3].

10.2 End-to-End Evaluation

We perform an end-to-end evaluation using the 200M-record OSMC dataset. In this experiment, the client continuously issues 100 queries in a pipeline manner to the server. We compare the total execution time (from the generation of the first query to the client receiving all responses) of Femur against the baselines. Femur is evaluated at full security and eight different security levels

[2]In our tests with a $2^{20} \times 256$ byte dataset, we observe that network communication, along with serialization and deserialization, account for more than 72% of the total execution time. Since the computational functionality of the Rust code is not the bottleneck, rewriting it in Java has minimal impact on the overall performance.

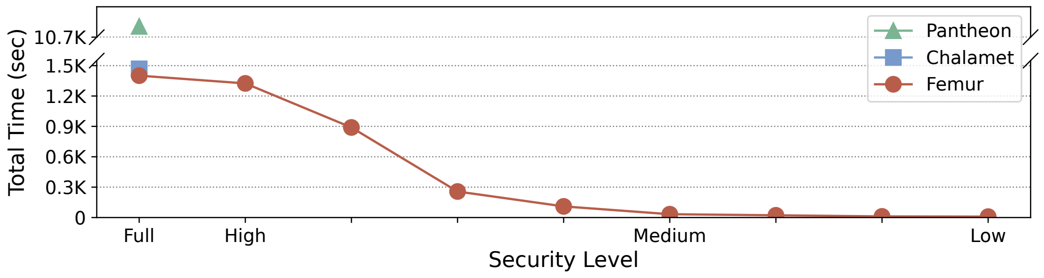


Fig. 4. Total Online Execution Time for 100 Queries – Femur uses 8 relaxed security levels besides full security. Pantheon and Chalamet encountered out-of-memory issues on 8 threads, so their results are ideally scaled from single-thread time. Pantheon failed to complete on this large dataset after running 24 hours using a single thread, so we mark its (lower-bound) time as 24/8 = 3 hours.

Table 1. Total Offline Time for 100 Queries (in seconds) – Femur only needs to be initialized once to support all security-level queries, so different t values produce similar offline latencies.

Pantheon	**Chalamet**	**Femur** With Relaxed Security Level (Various t)							
Full Security		1M	500K	100K	50K	10K	5K	1K	100
55.5	16214.3	697.5	697.2	703.2	697.3	698.2	694.6	702.1	696.2

with varying t values of 1M (high), 0.5M, 0.1M, 50K, 10K (medium), 5K, 1K, and 100 (low), which correspond to the average number of data points in Femur's computations of 256M[3], 128M, 25.6M, 12.8M, 2.56M, 1.28M, 256K, and 25.6K data points, respectively (plus an additional 129 data points generated by the PGM-index for each case).

We present the online execution time for Femur and the two baselines in Figure 4, and their offline execution time in Table 1. The results clearly show that relaxing the security significantly enhances the online runtime, with speedups of 1.11×, 1.65×, 5.71×, 13.4×, 44.6×, 66.8×, 131.1×, and 163.9× over Chalamet, which provides full security. Even at the same full security level, Femur is slightly faster than Chalamet (1.05×) and significantly outperforms Pantheon (over 7.71×). Femur reduces the average execution time per query from 14 seconds under full security to 89.6 ms at a relaxed security level that guarantees indistinguishability among 100 neighboring queries. This demonstrates an effective trade-off between security and performance, enabling significant speedups when users are willing to relax their privacy requirements.

Pantheon is extremely slow on large datasets, primarily due to its reliance on a slow homomorphic equality check during the online phase to locate the querying key. In contrast, Femur employs the PGM-index for rapid key location, at the slight cost of sending a small amount of data during initialization. Even with the default value of $\varepsilon_{\text{data}} = 64$, which is relatively small, the size of the PGM-index is only 5MB on the 200 million records. While Chalamet performs close to Femur with full security level, it must transfer 858.5 MB of ciphertext results, which can significantly delay query responses in bandwidth-constrained scenarios. Unlike Pantheon and Chalamet, whose computation or transmission times grow with dataset size, Femur maintains a constant online

[3]Note that this is a theoretical expectation. Since the OSMC dataset contains only 200M keys, any obfuscated range exceeding 200M will be capped at 200M.

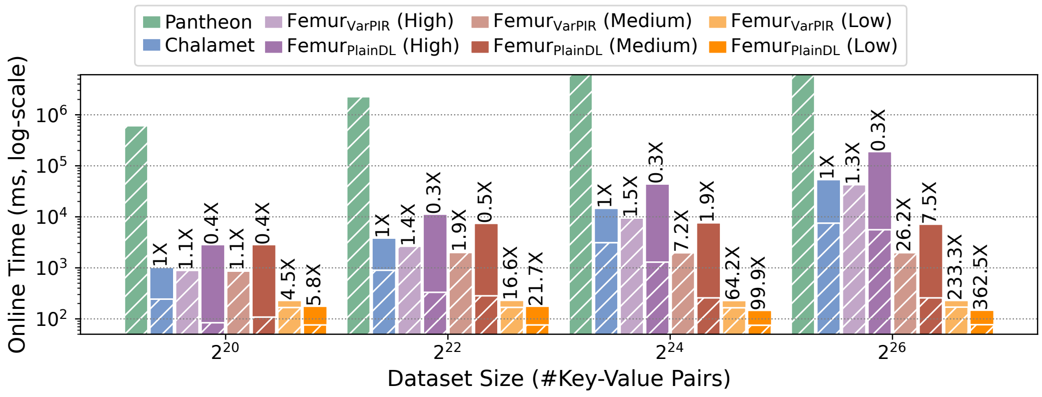


Fig. 5. Online Execution Time for Each Query on Different Dataset Sizes – The number above each bar represents the speedup over Chalamet. Pantheon timed out on datasets of size 2^{24} and 2^{26}. The shadowed area represents the server-side computation time, including the time required to serialize the data to be sent.

transfer time, limited to two ciphertexts (for query and response), with computation time dependent solely on the user-specified security level.

Chalamet suffers from a significantly slower offline phase, requiring 4.5 hours to pre-process 200 million key-value pairs as shown in Table 1. This makes Chalamet impractical in scenarios where updates or inserts are needed, as even batch updates require re-processing all data points, leading to substantial delays. In contrast, Femur completes the offline phase in 11.5 minutes, significantly faster than Chalamet. Besides, our update strategies in Section 9 further reduce the re-initialization time by minimizing the number of key-value pairs that need to be re-encoded. Note that to support the flexibility of different privacy levels in Femur, we do *not* need to re-execute the initialization phase. Pantheon's offline initialization time is short because it only requires the user to upload the two parameters for the underlying homomorphic encryption scheme. However, its online query processing time is significantly longer.

10.3 Impact of Various Factors

Starting from this section, we focus on the performance breakdown of individual queries and the impact of various factors on performance. Therefore, each query is run individually with a single thread, and the online execution time reflects the performance of a single query. Unless otherwise stated, other parameters remain the same as previously described.

10.3.1 Impact of Dataset Sizes. In this section, we evaluate the performance of each scheme across various dataset sizes. Specifically, we conduct experiments on a subset of the OSMC dataset of sizes 2^{20}, 2^{22}, 2^{24}, and 2^{26}. We also test smaller datasets, which are not included here due to space limitations, and observed conclusions consistent with the results presented. We select three security levels for Femur, corresponding to the expected length of obfuscated ranges of 256 million, 2.56 million, and 25600, to evaluate both Femur $_{\text{PlainDL}}$ and Femur $_{\text{VarPIR}}$ schemes. If the expected length exceeds the total number of data points in the dataset, the current scheme is the same as full security (i.e., high level for all, and medium level for 2^{20}).

Figure 5 illustrates the online time for each method across various dataset sizes, which mainly consists of the server-side computation time and the communication time to transmit data between the two parties. The online latency of Chalamet increases linearly with dataset size, rising from 1.02

Table 2. Online Communication per Query (in MB) – including the query keys/parameters uploaded by the client and the query results downloaded from the server.

Dataset Size	Baselines on Full Security		**Femur** on High Security		**Femur** on Medium Security		**Femur** on Low Security	
	Pantheon	**Chalamet**	**VarPIR**	**PlainDL**	**VarPIR**	**PlainDL**	**VarPIR**	**PlainDL**
2^{20}	3.5	4.5	0.43	16	0.43	16	0.43	0.39
2^{22}	3.5	18	0.43	64	0.43	39	0.43	0.39
2^{24}	3.5	72	0.43	256	0.43	39	0.43	0.39
2^{26}	3.5	288	0.43	1024	0.43	39	0.43	0.39

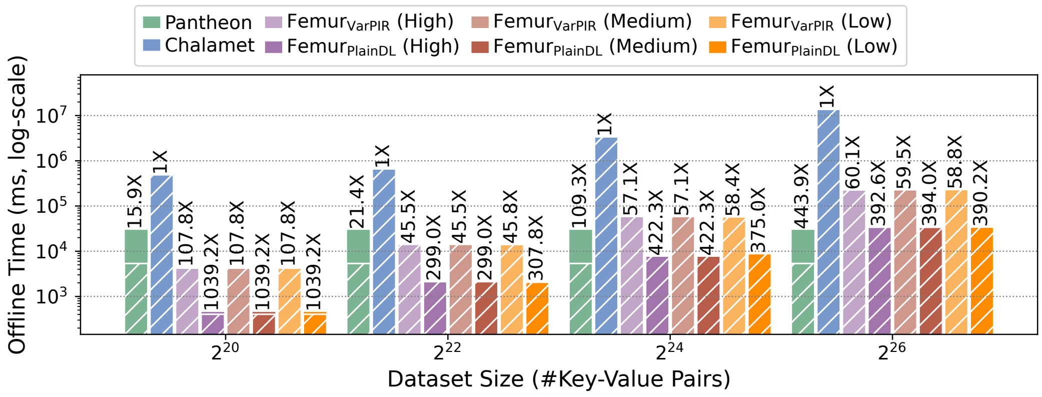


Fig. 6. Offline Time for Each Query on Different Dataset Sizes – The number above each bar is the speedup ratio of each scheme compared to Chalamet.

seconds for 2^{20} key-value pairs to 53.7 seconds for 2^{26} key-value pairs. This linear growth is not only due to the increased computation but also due to the linear increase in online communication, as shown in Table 2. Pantheon shows a significant computation time despite constant communication volumes (3.5 MB), with one query on 2^{22} entries taking an impractical 2,261 seconds. In contrast, Femur's online response time remains independent of dataset size at fixed security granularity. This is because the security requirements for the client typically do not escalate with the dataset size. Compared to Chalamet, Femur achieves a maximum speedup of 362.5× by relaxing security to ensure distance-based indistinguishability with $t = 100$ and $\epsilon_{\text{dp}} = 2^{-6}$. Furthermore, the online communication cost of Femur $_{\text{VarPIR}}$ is fixed at 0.43 MB (i.e., the size of a ciphertext), regardless of dataset size or security granularity. Similarly, the online communication cost of Femur $_{\text{PlainDL}}$ does not increase with dataset size for the same security levels, remaining at 0.39 MB for all low-security experiments. Note that for higher security levels in Table 2, the expected length exceeds the size of certain datasets, causing the online communication to degrade to the entire dataset size.

Among our two schemes, Femur $_{\text{PlainDL}}$ scheme is more sensitive to bandwidth. It does not involve plaintext/ciphertext computations, so its query time is determined by the bandwidth and the byte size of the data. In contrast, Femur $_{\text{VarPIR}}$ is compute-intensive. Owing to our encoding design, the network transmission involves only uploading a query and downloading a result, whose time is minimized and remains constant at 63 ms. Consequently, its overall runtime is almost dominated by computation. In Femur, the cost model automatically selects the faster scheme between the two.

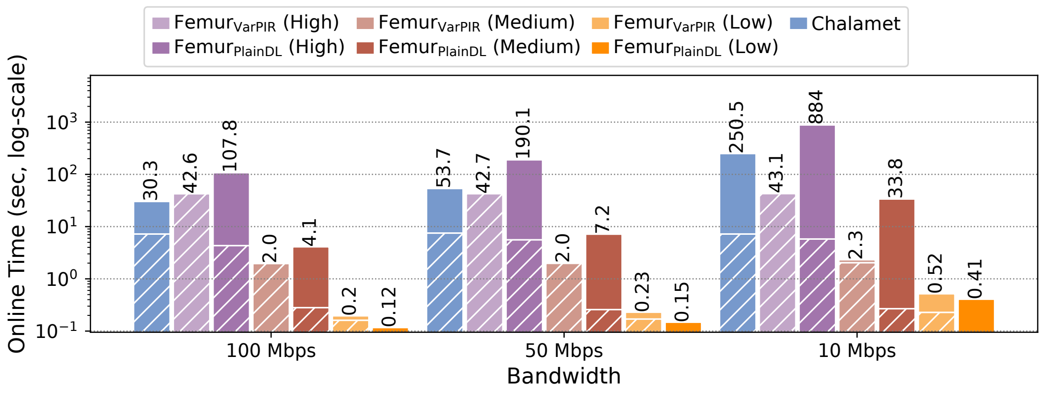


Fig. 7. Online Execution Time per Query at Various Bandwidths – The number above each bar is the latency.

Particularly, Chalamet and Femur $_{\text{VarPIR}}$ (both High and Medium) on the dataset size with 2^{20} provide comparable online end-to-end latency, with the same level of full security. However, in the offline phase, our Femur $_{\text{VarPIR}}$ offers a 107.8× speedup compared to Chalamet, as shown in Figure 6, by only encoding the records into FHE plaintexts and efficiently building and transmitting the PGM-index. Since the shorter offline phase of Femur $_{\text{PlainDL}}$ only accomplishes tasks that are a subset of Femur $_{\text{VarPIR}}$ (i.e., transmit PGM-index), Femur's overall offline time is determined by Femur $_{\text{VarPIR}}$. Notably, our offline time remains constant across different security levels, as it depends solely on the size of the public key-value store. Clients download the PGM-index once and can then use Femur for queries with different security requirements, whereas Chalamet and Pantheon rigidly support only fully secure queries.

10.3.2 Impact of Bandwidth. Figure 7 illustrates the online latency across different bandwidths for the dataset of 2^{26} key-value pairs. We evaluate three security levels of Femur: high, medium, and low. The high level aligns with the full security level of the baselines, as the expected length of the obfuscated range exceeds 2^{26}. Bandwidth has less impact on Femur $_{\text{VarPIR}}$ compared to both Femur $_{\text{PlainDL}}$ and Chalamet. Femur $_{\text{PlainDL}}$ is bandwidth-intensive and becomes faster with higher bandwidth. Chalamet, which transfers 288 MB of data, is similarly limited by bandwidth. At 10 Mbps, it takes 250.5 seconds to respond to a query. In contrast, Femur $_{\text{VarPIR}}$ is able to transfer only one ciphertext due to its misaligned encoding, and it is bound by the FHE computations rather than data transfers.

In summary, Femur, leveraging our cost model, adapts to various network environments and automatically selects the optimal solution between Femur $_{\text{VarPIR}}$ and Femur $_{\text{PlainDL}}$. Femur achieves response times ranging from 0.12 to 42.6 seconds, depending on the available bandwidth and the specified security level.

10.3.3 Impact of Value Size. Figure 8 illustrates the changes in both online and offline execution times for each scheme as the byte length of each value varies. The dataset consists of 2^{20} key-value pairs, with fixed 8-byte keys and the low security level for Femur.

The online execution time of Femur increases as the value length grows, but it continues to show advantages. This increase is due to the larger total byte size, which leads to more encoded plaintexts and also slows down the plaintext encoding process during the offline phase. When the value length is 8 bytes, Femur uses the PlainDL scheme. As the values become larger, PlainDL's performance degrades rapidly with more transmitted data. Thus, Femur switches to the VarPIR

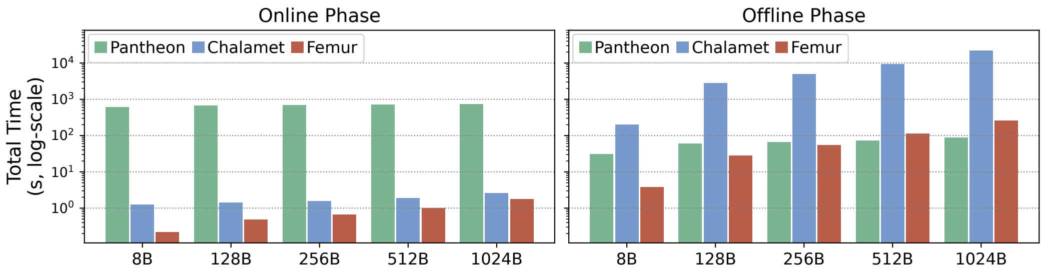


Fig. 8. Online/Offline Execution Time at Various Value Size (in bytes) – Both graphs share the same y-axis.

Table 3. Average Length of Obfuscated Ranges with Various Security Levels

Distance (t)	**10**	**100**	**1000**	**10000**
B+tree	65541.46	65552.8	262088.2	2621408.6
PGM-index	2715.12	25740.2	256135.1	2559313.1
Ratio	24.1394×	2.5461×	1.0233×	1.0243×

scheme at 128-byte values. Chalamet's online time increases slightly with the value length, but its offline time grows significantly, from 201.7 seconds to 6.02 hours. This is due to the rapid growth in computational cost caused by the larger key-value pair length, with most of the computation shifted to the offline phase. In contrast, Pantheon's bottleneck lies in the equality check between the querying key and all keys. Since the key length remains constant, its online processing time increases only slightly, from 609 seconds to 729 seconds.

10.3.4 Comparison of Client-Side Indexes. To demonstrate the benefits of using PGM-indexes, we compare PGM-indexes and B+trees based on the length of the obfuscated ranges generated and the index size. The obfuscated range length directly impacts the data volume downloaded or processed on the server. The index size determines the amount of data required for clients to download from the server during the offline phase, which is also crucial especially in scenarios with update operations.

We first implement the noise generation algorithm described in Section 7 on the B+tree and evaluate both indexes under various security levels. For each index, we process 10 million queries, converting querying keys into obfuscated ranges and measuring the average length of the resulting ranges. As shown in Table 3, when the security requirements are low, the obfuscated ranges generated by the B+tree are 24.14× longer than those generated by the PGM-index. Even as the security requirements increase, the PGM-index consistently reduces approximately 60,000 injected noise. These results demonstrate that the PGM-index, which operates at item-level granularity, is effective in minimizing injected noise while satisfying the same security requirements.

To evaluate the effects of data distribution and dataset size on index sizes, we construct both indexes using three datasets from the SOSD benchmark [58], each containing 200 million key-value pairs (two real-world datasets and one synthetic dataset following a normal distribution). Besides, we test index sizes on OSMC datasets of varying sizes. For the B+tree, only internal nodes are counted, and for the PGM-index, we measure sizes across different values of $\varepsilon_{\text{data}}$. As shown in Table 4, the PGM-index is significantly smaller than the B+tree, even with small $\varepsilon_{\text{data}}$ and on the OSMC dataset that is less friendly to learned indexes. Based on these results, we set the default

Table 4. Index sizes of B+trees and PGM-indexes (in MiB) – The bolded numbers are the index sizes used in our previous experiments, indicating the network required to transmit the index.

Size	**Dataset**	**B+tree**	**PGM-index (With Various $\varepsilon_{\text{data}}$)**				
			512	256	128	64	32
200 million	Wiki	11.93	0.11	0.17	0.29	0.58	1.31
	OSMC	11.93	0.65	1.28	2.55	**5.13**	10.3
	Normal	11.93	0.01	0.01	0.01	0.01	0.02
2^{20}	OSMC	0.13	0.01	0.01	0.02	**0.03**	0.07
2^{22}	OSMC	0.50	0.01	0.03	0.06	**0.11**	0.22
2^{24}	OSMC	2.00	0.06	0.12	0.24	**0.47**	0.93
2^{26}	OSMC	8.01	0.25	0.50	0.98	**1.95**	3.91

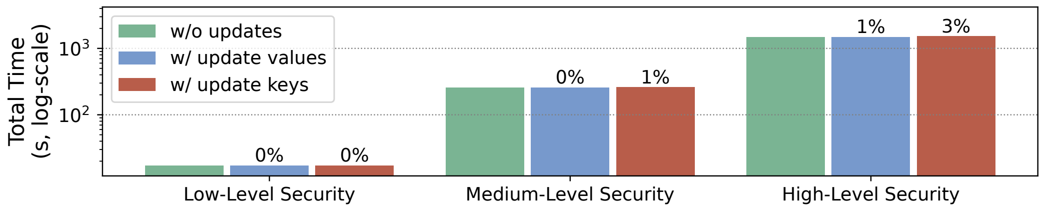


Fig. 9. Online Execution Time for 100 Queries on Redis and with Updates – Each number above the bar is the ratio of the additional time caused by updates to the time without updates.

value of $\varepsilon_{\text{data}}$ to 64 to balance the index size and the length of the predicted range. Meanwhile, $\varepsilon_{\text{model}}$ has a smaller impact on the index size, so we adopt the default value [40] of 4.

10.4 Evaluation on Redis

To evaluate the performance and usability of Femur in real-world scenarios, we implemented Femur on Redis, a widely used key-value store, using Jedis (the Redis Java client) [4]. The server also continuously updates its key-value pairs while serving private queries from clients. During lookup and update operations, we use the Jedis interface to retrieve data within the specified range and write updated plaintext back to Redis. To ensure uninterrupted client queries during updates, we employ Multi-Version Concurrency Control (MVCC) to manage multiple Redis instances, as detailed in Section 9. We conduct experiments by executing 100 lookup queries on this Redis-based version of Femur under three scenarios: (1) no updates, (2) real-time updates for values with a 1-second interval between updates, and (3) periodic batch updates for keys with a 1-second delay between completing one update and initiating the next. The dataset size is 2^{24}, and queries are performed at three security levels: low, medium, and high.

As shown in Figure 9, updates cause at most a 3% increase in total query time. This minor delay arises because some queries require fetching the updated PGM-index after a version switch, ensuring that all lookup results remain up-to-date. We also track the number of update operations performed during the execution of 100 queries. In the high-security evaluation, the server completes 1,120 real-time value updates and 24 periodic batch updates for keys/databases. The real-time updates take only 70 ms each, including reading plaintext from Redis, updating it, and writing it back to Redis. A periodic batch update takes approximately 60 seconds, however, we use MVCC to further reduce the impact of update operations by performing most tasks asynchronously in background threads. Compared to Chalamet, which requires 1.5 hours to reinitialize a database of the same size, Femur significantly reduces update cycles, making it more practical for real-world scenarios.

11 Related Work

Relaxed Security in Query Processing. Several works [19, 34, 39, 41, 56, 71, 72, 80, 82, 93, 94] have applied differential privacy (DP) to data management to provide relaxed security models with theoretical guarantees for secure query processing, thus enhancing performance. Specifically, Shrinkwrap [19] introduces DP-based noise to intermediate results, concealing the true size of the intermediate data and improving overall performance by eliminating the need for worst-case padding after each operator's execution. Adore [71] and Doquet [72] address access pattern leakage during relational operations and join queries using differentially oblivious operators and private data structures. Longshot [94] tackles the challenges of indexing a growing database by combining DP with secure multiparty computation (MPC). While prior work primarily targets scenarios involving online analytical processing (OLAP) or encrypted data, Femur enables private queries on public key-value stores, offering relaxed security with theoretical guarantees, ensuring that each query is indistinguishable from its neighbors.

Privacy Enhancements in DBMS. In recent years, significant efforts have focused on enabling clients to outsource private data while ensuring secure query processing [16, 17, 37, 68, 69, 74, 79]. These approaches typically involve encrypting data, storing it on a cloud server, and using techniques such as order-preserving encryption [6], homomorphic encryption [67, 79, 83], searchable encryption [24] and Trusted Execution Environments (TEE) [18, 20, 30, 75, 76, 81, 96] to achieve this goal. In contrast, Femur is designed for public datasets, which focuses on query privacy. Many works align with our goal [8, 10, 13, 59, 60, 62, 66, 95], mainly based on homomorphic encryption. For example, Pantheon [7] and Constant-weight PIR [57] implement equality-checking operators and invoke traditional PIR schemes for homomorphic operations. While our framework supports similar homomorphic operations to return target key-value pairs, it also offers the flexibility to utilize other schemes, such as direct downloads when bandwidth is sufficient. Moreover, existing PIR schemes typically require all data to participate in computation or communication, resulting in poor scalability and limited support for large datasets. However, our scheme allows for flexibility in adjusting the range of data participation in computations without requiring re-encoding. This works seamlessly with our definition of distance-based indistinguishability. By offering users the ability to relax security and select different levels of security guarantees based on their needs, Femur significantly accelerates query response, making Femur a more practical solution.

12 Conclusion

We present Femur, a framework that enables users to perform secure queries on public key-value stores, while empowering users to tailor privacy protections to their specific requirements. By employing the novel concept of distance-based indistinguishability and an adaptive retrieval mechanism supporting both direct downloads and an enhanced PIR scheme, Femur achieves a fine balance between privacy and performance. Our evaluations confirm that Femur efficiently supports a wide range of security configurations while maintaining practical query response times, even on large datasets.

Acknowledgments

This work was partially supported by the Shanghai Qi Zhi Institute Innovation Program (SQZ202406 & SQZ202314) and the National Social Science Foundation of China (Grant No. 22 & ZD147).

References

[1] 2009. Redis. https://redis.io/

[2] 2024. Chalamet. https://github.com/claucece/chalamet.

[3] 2024. Femur. https://github.com/alibaba-edu/mpc4j.

[4] 2024. Jedis. https://github.com/redis/jedis.

[5] Daniel J. Abadi, Samuel Madden, and Miguel Ferreira. 2006. Integrating compression and execution in column-oriented database systems. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, June 27-29, 2006*, Surajit Chaudhuri, Vagelis Hristidis, and Neoklis Polyzotis (Eds.). ACM, 671–682. doi:10.1145/1142473.1142548

[6] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. 2004. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data* (Paris, France) *(SIGMOD '04)*. Association for Computing Machinery, New York, NY, USA, 563–574. doi:10.1145/1007568.1007632

[7] Ishtiyaque Ahmad, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2022. Pantheon: Private Retrieval from Public Key-Value Store. *Proc. VLDB Endow.* 16, 4 (dec 2022), 643–656. doi:10.14778/3574245.3574251

[8] Ishtiyaque Ahmad, Yuntian Yang, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. 2021. Addra: Metadata-private voice communication over fully untrusted infrastructure. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*.

[9] Ahmet Aktay, Shailesh Bavadekar, Gwen Cossoul, John Davis, Damien Desfontaines, Alex Fabrikant, Evgeniy Gabrilovich, Krishna Gadepalli, Bryant Gipson, Miguel Guevara, et al. 2020. Google COVID-19 community mobility reports: anonymization process description (version 1.1). *arXiv preprint arXiv:2004.04145* (2020).

[10] Asra Ali, Tancrède Lepoint, Sarvar Patel, Mariana Raykova, Phillipp Schoppmann, Karn Seth, and Kevin Yeo. 2021. {Communication–Computation} trade-offs in {PIR}. In *30th USENIX security symposium (USENIX Security 21)*. 1811–1828.

[11] Mário Alvim, Konstantinos Chatzikokolakis, Catuscia Palamidessi, and Anna Pazii. 2018. Invited Paper: Local Differential Privacy on Metric Spaces: Optimizing the Trade-Off with Utility. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. 262–267. doi:10.1109/CSF.2018.00026

[12] Kareem Amin, Jennifer Gillenwater, Matthew Joseph, Alex Kulesza, and Sergei Vassilvitskii. 2022. Plume: differential privacy at scale. *arXiv preprint arXiv:2201.11603* (2022).

[13] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with compressed queries and amortized query processing. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 962–979.

[14] Sebastian Angel and Srinath Setty. 2016. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 551–569.

[15] Christoph Anneser, Andreas Kipf, Huanchen Zhang, Thomas Neumann, and Alfons Kemper. 2022. Adaptive Hybrid Indexes. In *SIGMOD '22: International Conference on Management of Data, Philadelphia, PA, USA, June 12 - 17, 2022*, Zachary G. Ives, Angela Bonifati, and Amr El Abbadi (Eds.). ACM, 1626–1639. doi:10.1145/3514221.3526121

[16] Panagiotis Antonopoulos, Arvind Arasu, Kunal D. Singh, Ken Eguro, Nitish Gupta, Rajat Jain, Raghav Kaushik, Hanuma Kodavalla, Donald Kossmann, Nikolas Ogg, Ravi Ramamurthy, Jakub Szymaszek, Jeffrey Trimmer, Kapil Vaswani, Ramarathnam Venkatesan, and Mike Zwilling. 2020. Azure SQL Database Always Encrypted. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 1511–1525. doi:10.1145/3318464.3386141

[17] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase.. In *CIDR*.

[18] Maurice Bailleu, Jörg Thalheim, Pramod Bhatotia, Christof Fetzer, Michio Honda, and Kapil Vaswani. 2019. Speicher: securing LSM-based key-value stores using shielded execution. In *Proceedings of the 17th USENIX Conference on File and Storage Technologies* (Boston, MA, USA) *(FAST'19)*. USENIX Association, USA, 173–190.

[19] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient SQL query processing in differentially private data federations. *Proc. VLDB Endow.* 12, 3 (Nov. 2018), 307–320. doi:10.14778/3291264.3291274

[20] Ilaria Battiston, Lotte Felius, Sam Ansmink, Laurens Kuiper, and Peter Boncz. 2024. DuckDB-SGX2: The Good, The Bad and The Ugly within Confidential Analytical Query Processing. In *Proceedings of the 20th International Workshop on Data Management on New Hardware* (Santiago, AA, Chile) *(DaMoN '24)*. Association for Computing Machinery, New York, NY, USA, Article 14, 5 pages. doi:10.1145/3662010.3663447

[21] Rudolf Bayer. 1972. Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms. *Acta informatica* 1, 4 (dec 1972), 290–306. doi:10.1007/BF00289509

[22] Nikita Borisov, George Danezis, and Ian Goldberg. 2015. DP5: A private presence service. *Proceedings on Privacy Enhancing Technologies* (2015).

[23] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2014. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)* 6, 3 (2014), 1–36.

[24] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. 2014. Dynamic searchable encryption in very-large databases: Data structures and implementation. *Cryptology ePrint Archive* (2014).

[25] Sofía Celi and Alex Davidson. 2024. Call Me By My Name: Simple, Practical Private Information Retrieval for Keyword Queries. *Cryptology ePrint Archive* (2024).

[26] Konstantinos Chatzikokolakis, Miguel E. Andrés, Nicolás Emilio Bordenabe, and Catuscia Palamidessi. 2013. Broadening the Scope of Differential Privacy Using Metrics. In *Privacy Enhancing Technologies*, Emiliano De Cristofaro and Matthew Wright (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 82–102.

[27] Hao Chen, Kim Laine, and Rachel Player. 2017. Simple encrypted arithmetic library-SEAL v2. 1. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 3–18.

[28] Benny Chor, Niv Gilboa, and Moni Naor. 1997. Private information retrieval by keywords. (1997).

[29] Henry Corrigan-Gibbs and Dmitry Kogan. 2020. Private information retrieval with sublinear online time. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*. Springer, 44–75.

[30] Victor Costan. 2016. Intel SGX explained. *IACR Cryptol, EPrint Arch* (2016).

[31] Alex Davidson, Gonçalo Pestana, and Sofía Celi. 2023. Frodopir: Simple, scalable, single-server private information retrieval. *Proceedings on Privacy Enhancing Technologies* (2023).

[32] Daniel Demmler, Peter Rindal, Mike Rosulek, and Ni Trieu. 2018. PIR-PSI: scaling private contact discovery. *Proceedings on Privacy Enhancing Technologies* (2018).

[33] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 969–984. doi:10.1145/3318464.3389711

[34] Wei Dong, Dajun Sun, and Ke Yi. 2023. Better than Composition: How to Answer Multiple Relational Queries under Differential Privacy. *Proc. ACM Manag. Data* 1, 2, Article 123 (June 2023), 26 pages. doi:10.1145/3589268

[35] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.

[36] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (Scottsdale, Arizona, USA) *(CCS '14)*. Association for Computing Machinery, New York, NY, USA, 1054–1067. doi:10.1145/2660267.2660348

[37] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: oblivious query processing for secure databases. *Proc. VLDB Endow.* 13, 2 (Oct. 2019), 169–183. doi:10.14778/3364324.3364331

[38] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive* (2012).

[39] Juanru Fang and Ke Yi. 2024. Privacy Amplification by Sampling under User-level Differential Privacy. *Proc. ACM Manag. Data* 2, 1, Article 34 (March 2024), 26 pages. doi:10.1145/3639289

[40] Paolo Ferragina and Giorgio Vinciguerra. 2020. The PGM-Index: A Fully-Dynamic Compressed Learned Index with Provable Worst-Case Bounds. *Proceedings of the VLDB Endowment* 13, 8 (apr 2020), 1162–1175. doi:10.14778/3389133.3389135

[41] Congcong Fu, Hui Li, Jian Lou, Huizhen Li, and Jiangtao Cui. 2023. DP-starJ: A Differential Private Scheme towards Analytical Star-Join Queries. *Proc. ACM Manag. Data* 1, 4, Article 238 (Dec. 2023), 24 pages. doi:10.1145/3626725

[42] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[43] Alexander Heinrich, Matthias Hollick, Thomas Schneider, Milan Stute, and Christian Weinert. 2021. PrivateDrop: Practical Privacy-Preserving Authentication for Apple AirDrop. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3577–3594. https://www.usenix.org/conference/usenixsecurity21/presentation/heinrich

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

[45] 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 23)*. USENIX Association, Anaheim, CA, 3889–3905. https://www.usenix.org/conference/usenixsecurity23/presentation/henzinger

[46] Noah Johnson, Joseph P. Near, Joseph M. Hellerstein, and Dawn Song. 2020. Chorus: a Programming Framework for Building Scalable Differential Privacy Mechanisms. In *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. 535–551. doi:10.1109/EuroSP48549.2020.00041

[47] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proc. VLDB Endow.* 11, 5 (jan 2018), 526–539. doi:10.1145/3187009.3177733

[48] Marcel Keller, Emmanuela Orsini, and Peter Scholl. 2016. MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 830–842. doi:10.1145/2976749.2978357

[49] Marcel Keller, Valerio Pastro, and Dragos Rotaru. 2018. Overdrive: Making SPDZ great again. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 158–189.

[50] Dmitry Kogan and Henry Corrigan-Gibbs. 2021. Private blocklist lookups with checklist. In *30th USENIX security symposium (USENIX Security 21)*. 875–892.

[51] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *Proceedings of the 2018 International Conference on Management of Data* (Houston, TX, USA) *(SIGMOD '18)*. Association for Computing Machinery, New York, NY, USA, 489–504. doi:10.1145/3183713.3196909

[52] Albert Hyukjae Kwon, David Lazar, Srinivas Devadas, and Bryan Ford. 2015. Riffle: An efficient communication system with strong anonymity. (2015).

[53] Hai Lan, Zhifeng Bao, J. Shane Culpepper, and Renata Borovica-Gajic. 2023. Updatable Learned Indexes Meet Disk-Resident DBMS - From Evaluations to Design Choices. *Proceedings of the ACM on Management of Data* 1, 2, Article 139 (jun 2023), 22 pages. doi:10.1145/3589284

[54] Xiaochen Li, Yuke Hu, Weiran Liu, Hanwen Feng, Li Peng, Yuan Hong, Kui Ren, and Zhan Qin. 2022. OpBoost: a vertical federated tree boosting framework based on order-preserving desensitization. *Proc. VLDB Endow.* 16, 2 (Oct. 2022), 202–215. doi:10.14778/3565816.3565823

[55] Yihao Liu, Xinyu Zeng, and Huanchen Zhang. 2024. LeCo: Lightweight Compression via Learning Serial Correlations. *Proc. ACM Manag. Data* 2, 1 (2024), 65:1–65:28. doi:10.1145/3639320

[56] Qiyao Luo, Yilei Wang, Ke Yi, Sheng Wang, and Feifei Li. 2023. Secure Sampling for Approximate Multi-party Query Processing. *Proceedings of the ACM on Management of Data* 1, 3 (2023), 219:1–219:27.

[57] 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 22)*. USENIX Association, Boston, MA, 1723–1740. https://www.usenix.org/conference/usenixsecurity22/presentation/mahdavi

[58] Ryan Marcus, Andreas Kipf, Alexander van Renen, Mihail Stoian, Sanchit Misra, Alfons Kemper, Thomas Neumann, and Tim Kraska. 2020. Benchmarking Learned Indexes. *Proceedings of the VLDB Endowment* 14, 1 (sep 2020), 1–13. doi:10.14778/3421424.3421425

[59] Carlos Aguilar Melchor, Joris Barrier, Laurent Fousse, and Marc-Olivier Killijian. 2016. XPIR: Private information retrieval for everyone. *Proceedings on Privacy Enhancing Technologies* (2016), 155–174.

[60] 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)*. IEEE, 930–947.

[61] Solomon Messing, Christina DeGregorio, Bennett Hillenbrand, Gary King, Saurav Mahanti, Zagreb Mukerjee, Chaya Nayak, Nate Persily, Bogdan State, and Arjun Wilkins. 2020. Facebook Privacy-Protected Full URLs Data Set. *Version DRAFT VERSION* (2020).

[62] Prateek Mittal, Femi Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. 2011. {PIR-Tor}: Scalable anonymous communication using private information retrieval. In *20th USENIX security symposium (USENIX security 11)*.

[63] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 35–52. doi:10.1145/3243734.3243760

[64] Muhammad Haris Mughees, Hao Chen, and Ling Ren. 2021. OnionPIR: Response efficient single-server PIR. In *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*. 2292–2306.

[65] Muhammad Haris Mughees and Ling Ren. 2023. Vectorized batch private information retrieval. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 437–452.

[66] Sarvar Patel, Joon Young Seo, and Kevin Yeo. 2023. Don't be dense: efficient keyword PIR for sparse databases. In *Proceedings of the 32nd USENIX Conference on Security Symposium* (Anaheim, CA, USA) *(SEC '23)*. USENIX Association, USA, Article 216, 18 pages.

[67] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: an encrypted database using semantically secure encryption. *Proc. VLDB Endow.* 12, 11 (July 2019), 1664–1678. doi:10.14778/3342263.3342641

[68] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles* (Cascais, Portugal) *(SOSP '11)*. Association for Computing Machinery, New York, NY, USA, 85–100. doi:10.1145/2043556.2043566

[69] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database Using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. 264–278. doi:10.1109/SP.2018.00025

[70] Yiming Qiao, Yihan Gao, and Huanchen Zhang. 2024. Blitzcrank: Fast Semantic Compression for In-memory Online Transaction Processing. *Proc. VLDB Endow.* 17, 10 (2024), 2528–2540. doi:10.14778/3675034.3675044

[71] Lianke Qin, Rajesh Jayaram, Elaine Shi, Zhao Song, Danyang Zhuo, and Shumo Chu. 2022. Adore: Differentially Oblivious Relational Database Operators. *Proc. VLDB Endow.* 16, 4 (Dec. 2022), 842–855. doi:10.14778/3574245.3574267

[72] Lina Qiu, Georgios Kellaris, Nikos Mamoulis, Kobbi Nissim, and George Kollios. 2023. Doquet: Differentially Oblivious Range and Join Queries with Private Data Structures. *Proc. VLDB Endow.* 16, 13 (Sept. 2023), 4160–4173. doi:10.14778/3625054.3625055

[73] SEAL 2022. Microsoft SEAL (release 4.1). https://github.com/Microsoft/SEAL. Microsoft Research, Redmond, WA..

[74] Mo Sha, Yifan Cai, Sheng Wang, Linh Thi Xuan Phan, Feifei Li, and Kian-Lee Tan. 2024. Object-oriented Unified Encrypted Memory Management for Heterogeneous Memory Architectures. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 155.

[75] Mo Sha, Jialin Li, Sheng Wang, Feifei Li, and Kian-Lee Tan. 2023. TEE-based General-purpose Computational Backend for Secure Delegated Data Processing. *Proceedings of the ACM on Management of Data* 1, 4 (2023), 263:1–263:28.

[76] Yuanyuan Sun, Sheng Wang, Huorong Li, and Feifei Li. 2021. Building enclave-native storage engines for practical encrypted databases. *Proc. VLDB Endow.* 14, 6 (Feb. 2021), 1019–1032. doi:10.14778/3447689.3447705

[77] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and Xiaofeng Wang. 2017. Privacy loss in apple's implementation of differential privacy on macos 10.12. *arXiv preprint arXiv:1709.02753* (2017).

[78] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, et al. 2019. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*. 1556–1571.

[79] Stephen Tu, M. Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing analytical queries over encrypted data. *Proc. VLDB Endow.* 6, 5 (March 2013), 289–300. doi:10.14778/2535573.2488336

[80] Leixia Wang, Qingqing Ye, Haibo Hu, and Xiaofeng Meng. 2024. PriPL-Tree: Accurate Range Query for Arbitrary Distribution under Local Differential Privacy. *Proc. VLDB Endow.* 17, 11 (Aug. 2024), 3031–3044. doi:10.14778/3681954.3681981

[81] Sheng Wang, Yiran Li, Huorong Li, Feifei Li, Chengjin Tian, Le Su, Yanshan Zhang, Yubing Ma, Lie Yan, Yuanyuan Sun, Xuntao Cheng, Xiaolong Xie, and Yu Zou. 2022. Operon: an encrypted database for ownership-preserving data management. *Proc. VLDB Endow.* 15, 12 (Aug. 2022), 3332–3345. doi:10.14778/3554821.3554826

[82] Yilei Wang, Xiangdong Zeng, Sheng Wang, and Feifei Li. 2025. Jodes: Efficient Oblivious Join in the Distributed Setting. In *Proceedings of the 51st International Conference on Very Large Data Bases (VLDB 2025)*. London, United Kingdom.

[83] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. 2014. Secure query processing with data interoperability in a cloud database environment. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1395–1406. doi:10.1145/2588555.2588572

[84] Chaichon Wongkham, Baotong Lu, Chris Liu, Zhicong Zhong, Eric Lo, and Tianzheng Wang. 2022. Are Updatable Learned Indexes Ready? *Proceedings of the VLDB Endowment* 15, 11 (jul 2022), 3004–3017. doi:10.14778/3551793.3551848

[85] David J Wu, Joe Zimmerman, Jérémy Planul, and John C Mitchell. 2016. Privacy-preserving shortest path computation. *arXiv preprint arXiv:1601.02281* (2016).

[86] Jiacheng Wu, Yong Zhang, Shimin Chen, Jin Wang, Yu Chen, and Chunxiao Xing. 2021. Updatable Learned Index with Precise Positions. *Proceedings of the VLDB Endowment* 14, 8 (apr 2021), 1276–1288. doi:10.14778/3457390.3457393

[87] Kevin Yeo. 2023. Lower bounds for (batch) PIR with private preprocessing. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 518–550.

[88] Xinyu Zeng, Yulong Hui, Jiahong Shen, Andrew Pavlo, Wes McKinney, and Huanchen Zhang. 2023. An Empirical Evaluation of Columnar Storage Formats. *Proc. VLDB Endow.* 17, 2 (2023), 148–161. doi:10.14778/3626292.3626298

[89] Huanchen Zhang, David G. Andersen, Andrew Pavlo, Michael Kaminsky, Lin Ma, and Rui Shen. 2016. Reducing the Storage Overhead of Main-Memory OLTP Databases with Hybrid Indexes. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 1567–1581. doi:10.1145/2882903.2915222

[90] Huanchen Zhang, Xiaoxuan Liu, David G. Andersen, Michael Kaminsky, Kimberly Keeton, and Andrew Pavlo. 2020. Order-Preserving Key Compression for In-Memory Search Trees. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19, 2020*, David Maier,

Rachel Pottinger, AnHai Doan, Wang-Chiew Tan, Abdussalam Alawini, and Hung Q. Ngo (Eds.). ACM, 1601–1615. doi:10.1145/3318464.3380583

[91] Jiaoyi Zhang and Yihan Gao. 2022. CARMI: A Cache-Aware Learned Index with a Cost-Based Construction Algorithm. *Proceedings of the VLDB Endowment* 15, 11 (jul 2022), 2679–2691. doi:10.14778/3551793.3551823

[92] Jiaoyi Zhang, Kai Su, and Huanchen Zhang. 2024. Making In-Memory Learned Indexes Efficient on Disk. *Proc. ACM Manag. Data* 2, 3, Article 151 (may 2024), 26 pages. doi:10.1145/3654954

[93] Shufan Zhang and Xi He. 2023. DProvDB: Differentially Private Query Processing with Multi-Analyst Provenance. *Proc. ACM Manag. Data* 1, 4, Article 267 (Dec. 2023), 27 pages. doi:10.1145/3626761

[94] Yanping Zhang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Longshot: Indexing Growing Databases Using MPC and Differential Privacy. *Proc. VLDB Endow.* 16, 8 (April 2023), 2005–2018. doi:10.14778/3594512.3594529

[95] Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. 2024. Piano: extremely simple, single-server PIR with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 4296–4314.

[96] Yu Zou, Yiran Li, Sheng Wang, Le Su, Zhen Gu, Yanheng Lu, Yijin Guan, Dimin Niu, Mingyu Gao, Yuan Xie, and Feifei Li. 2024. Salus: A Practical Trusted Execution Environment for CPU-FPGA Heterogeneous Cloud Platforms. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2024)*. San Diego, USA.

Received October 2024; revised January 2025; accepted February 2025