scientific reports

Check for updates

OPEN

A novel lightweight medical blockchain data query scheme

Yunzhen Zhu[1]✉, Xiaohong Deng[2,3]✉, Jiayan Liu[1], Yijie Zou[2], Juan Li[2] & Yuxin Fang[2]

To address the limitations of lightweight medical blockchains in terms of data query efficiency and nonexistence proofs, we propose a novel lightweight medical blockchain data query scheme. First, the XGBoost algorithm is employed to predict medical data weights, with high-weight data storage near the blockchain's root nodes being prioritized, thereby optimizing the storage architecture and enhancing query efficiency. Second, an efficient query method that combines aggregated Bloom filters and Merkle–Huffman (MH) trees is designed. Through segmented filtering and weight optimization, the query path length is reduced, improving the on-chain data query performance. Finally, to address the challenge of the data nonexistence proof, we propose a multi-node collaborative verification mechanism that integrates Bloom filters with a dynamic reputation system. By adaptively selecting high-credibility nodes and employing multi-node consensus, false positives are minimized, ensuring query accuracy and reliability. Theoretical analysis and simulation results show that, compared with existing schemes, the proposed approach improves the query efficiency by approximately 15%. Moreover, integrating multi-node collaborative verification with a dynamic reputation mechanism effectively mitigates malicious attack risk and enhances system security, making it particularly suitable for resource-constrained scenarios such as mobile health care.

Keywords Medical blockchain, Data query, Bloom filter, Merkle tree, Huffman tree

Medical data sharing is crucial for resource integration and for optimizing diagnosis and treatment, ultimately improving patient experience and health care quality[1]. As the core component of medical data sharing, data queries directly impact the efficiency and accuracy of the shared system. However, current medical data query systems suffer from poor interoperability[2], low efficiency[3], and privacy leaks[4]. With its decentralized architecture, distributed storage, and tamper-resistant properties, blockchain technology provides a novel solution that enhances both the performance[5] and security[6] of medical data queries.

Most existing blockchain-based data query solutions are integrated with cloud storage models, where data are encrypted and stored in the cloud, while only index information is maintained in the blockchain. For example, Samala et al.[7] proposed an electronic health record management system that enables data sharing and querying by storing data in the cloud and preserving index information in the blockchain. Kumari et al.[8] introduced the HealthRec-Chain framework, which integrates the Interplanetary File System (IPFS) with blockchain technology to increase data security and query efficiency. Li et al.[9] developed a blockchain-based distributed cloud storage system incorporating reliable deduplication mechanisms and storage balancing capabilities, which achieves sharded data distribution and load balancing through secret sharing and heuristic matching algorithms. However, while these advancements demonstrate notable progress in enhancing access efficiency for blockchain-based storage systems, the cloud storage implementation paradigm exhibits inherent limitations in system autonomy, degree of decentralization, and security assurance due to its reliance on third-party cloud resources.

To achieve greater decentralization and data autonomy, data can be directly stored in the blockchain. However, as data continuously accumulate in the form of blocks, synchronizing the entire blockchain at each node significantly increases the storage and bandwidth demands. To address this issue, researchers have proposed lightweight blockchain architectures[10], where resource-rich nodes act as full nodes, maintaining a complete blockchain backup, whereas resource-limited nodes function as light nodes, storing only block headers and other essential metadata. This approach effectively reduces storage pressure while maintaining decentralization. As a representative example, the EDCOMA framework proposed by Yu et al.[11] extends this foundation by integrating dual compression with zero-knowledge proofs, developing an efficient lightweight auditing mechanism that

[1]School of Information Engineering, Jiangxi University of Science and Technology, Ganzhou 341000, China. [2]School of Information Engineering, Gannan University of Science and Technology, Ganzhou 341000, China. [3]Key Laboratory of Cloud Computing and Big Data, Ganzhou 341000, China. ✉email: 6720230780@mail.jxust.edu.cn; dengxiaohong@gnust.edu.cn

effectively reduces resource consumption during the auditing process. This innovation demonstrates particular applicability in resource-constrained storage node environments.

While lightweight blockchain architectures reduce storage demands, a key challenge remains in further improving query efficiency and accuracy. To increase query efficiency, Yang et al.[12] proposed a verifiable query scheme based on an off-chain committee (CVCQ) to address cross-chain query challenges in large-scale blockchain-enabled IoT (BIoT) systems. By decoupling data queries and consensus processes into off-chain committee operations, this approach reduces consensus complexity while utilizing aggregated accumulators to minimize cross-chain data integrity verification overhead to constant level, thereby improving cross-regional query throughput. Nevertheless, the CVCQ scheme primarily targets IoT environments with geographical partitioning characteristics, requiring further validation regarding its adaptability and efficiency in medical blockchain scenarios. Li et al.[13] developed the FlexIM system, which employs reinforcement learning to dynamically select optimal index configurations based on query workload characteristics. Through innovative designs of Root Merkle Tree (RMT) and Bloom-filter Merkle Tree (BMT), the system reduces storage overhead while preserving verifiability, consequently improving query efficiency. However, its index benefit evaluation model shows insufficient consideration of dynamic weight allocation for data characteristics, and lacks specialized optimization for index coordination and verification in cross-chain query scenarios. Xiao et al.[14] introduced a two-phase distributed query framework with noise injection, leveraging blockchain's autonomous features to effectively conceal retrieval patterns while enhancing query efficiency. Yet, the multi-phase communication and reorganization overhead incurred from data partitioning and result aggregation may substantially increase system latency in large-scale or cross-chain query environments, potentially compromising efficiency gains. Zhang et al.[15] proposed a multilevel indexing method, which partitions the blockchain using a weight matrix. Within each partition, the master chain constructs indices via a skip-consistent hashing algorithm, whereas secondary indices are built on slave chains using Bloom filters and skip lists. However, this indexing approach requires additional storage space, and the index size directly affects the query speed. Jia et al.[16] proposed a three-layer model consisting of a data evaluation layer, a storage layer, and a distribution layer. By extracting medical data features and using the extreme learning machine method to predict and evaluate the weights of medical data, they achieved data storage and querying on the blockchain. However, this approach requires sequential traversal of blocks during queries, resulting in low query efficiency. Liu et al.[17] introduced a quad-Merkle tree structure combined with smart contracts to improve query efficiency. Nevertheless, this quadtree-based structure still encounters efficiency bottlenecks when handling large-scale medical data and high-frequency repeated queries. Moreover, this method still requires block-by-block traversal during queries, leading to suboptimal overall query efficiency.

Moreover, to ensure the accuracy of the query results, verification of the retrieved data is needed. Currently, there are two mainstream approaches for verifiable queries. One approach involves constructing a verifiable data structure on the basis of cryptographic accumulators. However, this method incurs high verification costs, and the large number of generated key pairs not only reduces the query speed but also increases the complexity of key management. The other approach uses the hash verification property of Merkle trees to construct verifiable data structures. For example, Sun et al.[18] designed a simplified Merkle-B tree structure that verifies the Merkle root by returning a validation path, thereby ensuring the existence and integrity of query results and preventing data tampering. However, when a query result is nonexistent, most existing methods overlook providing a nonexistence proof, allowing malicious nodes to launch denial-of-service attacks. To address this issue, Zhang et al.[19] proposed the introduction of the Merkle mountain range (MMR), where full nodes possess the latest valid blocks to light nodes, ensuring the trustworthiness of the query results. However, this approach requires proofs be generated for each query, leading to high communication overhead. With respect to proofs for nonexistent query results, most existing methods assume fully trusted nodes and commonly adopt sorted Merkle trees. For example, Cheng et al.[20] first sorted the elements before storing the data in leaf nodes. To prove the nonexistence of a specific element, the method must demonstrate that the element does not exist between two adjacent existing elements. However, this proof method is restricted to sorted structures and requires high storage costs for path proofs, making it unsuitable for scenarios with high complexity and frequent repetition in medical data queries.

To address the issues of low query efficiency in lightweight medical blockchain systems and the limitations of nonexistence proofs in sorted Merkle trees, this paper proposes a novel lightweight medical blockchain data query method. The main contributions of this study are as follows:

1. To address imbalanced query frequencies and inefficient retrieval of high-frequency medical data, we propose a medical data storage scheme based on weight prediction. By leveraging the XGBoost algorithm to predict data access weights and optimize the blockchain storage structure, high-weight data are stored closer to the root node with priority, improving retrieval efficiency and reducing query overhead.
2. To increase the query efficiency and eliminate the need for block-by-block searches, we introduce a query method that combines aggregated Bloom filters and a Merkle Huffman (MH) tree. By segmenting blocks and applying an aggregated Bloom filter matrix within each segment, irrelevant blocks are rapidly filtered out during queries. For intrablock searches, the MH tree structure places higher-weight nodes closer to the root, increasing the query speed.
3. To overcome the limitations of sorted Merkle trees in proving data nonexistence, we propose a collaborative multinode verification method that integrates Bloom filters with a reputation mechanism. This method dynamically selects high-trust nodes for queries on the basis of a reputation model and directly generates nonexistence proofs using Bloom filters, ensuring efficient verification. When false positives occur in the Bloom filter, multinode collaborative verification is employed to reach consensus, further enhancing the trustworthiness of the query results.

The remainder of this paper is structured as follows: Section 2 "Problem statement" presents the problem statement, provides a detailed discussion of the challenges in lightweight medical blockchain data queries and highlights the necessity of this research. Section 3 "Weight prediction-based lightweight medical blockchain storage architecture" systematically elaborates on the proposed lightweight medical blockchain architecture on the basis of weight prediction. Section 4 "Data query methods and nonexistence proofs" introduces the specific query method that integrates aggregated Bloom filters and the MH tree, along with the multinode verification mechanism based on reputation. Section 5 "Feasibility analysis of the proposed scheme" conducts theoretical and simulation-based analyses of the proposed scheme and demonstrates its advantages by comparing it with other existing solutions. Finally, the paper concludes with a summary in Section 6 "Conclusion".

Problem statement

Low efficiency of on-chain data queries

In blockchain systems, blocks consist of a block header and block body. The chain structure inherently limits data query efficiency, which can be categorized into interblock and intrablock searches.

Interblock search process

During the interblock query process, not all blocks contain relevant query results. To improve query efficiency and avoid unnecessary intrablock queries in irrelevant blocks, the Bloom filter (BF)[21], an efficient data structure, can be utilized. As shown in Fig. 1, the Bloom filter-based block-by-block search process consists of a binary bit array of length m and k hash functions. When an element is inserted, the process uses k hash functions to map the element to k index positions ranging from 1 to m and sets the corresponding k positions to 1 to indicate the element's presence. During querying, the system checks the bit array using the same k hash calculations. If any of these positions is 0, the element is definitively not in the set. If all the positions are 1, the element is likely present, but false positives may occur.

As shown in the Bloom filter on the right side of Fig. 1, assuming that $k=3$ and $m=16$, element e_1 is in set S, whereas elements e_2 and e_3 are not in the set. After three hash calculations are performed for elements e_1, e_2 and e_3, all corresponding index positions of e_1 are set to 1, indicating that the element is present in the set. Since the corresponding index positions of e_2 are not all 1, it can be conclusively proven that the element is not in the set. After three hash calculations are performed for e_3, the corresponding index positions in this Bloom filter 'coincidentally' all become 1, although e_3 is actually not present in the set. This type of error caused by hash collisions is termed a false-positive.

By adding a Bloom filter field in the block header, all attribute values within the block can be added to the Bloom filter. During subsequent query processes, the block's potential inclusion of query results can be verified by sequentially comparing Bloom filters in block headers. All corresponding index positions in the Bloom filter will be 1 only if valid data exist or a false-positive occurs. In such cases, an intrablock query must be performed. Although Bloom filters inherently exhibit false positives, the error rate can be reduced by adjusting the filter

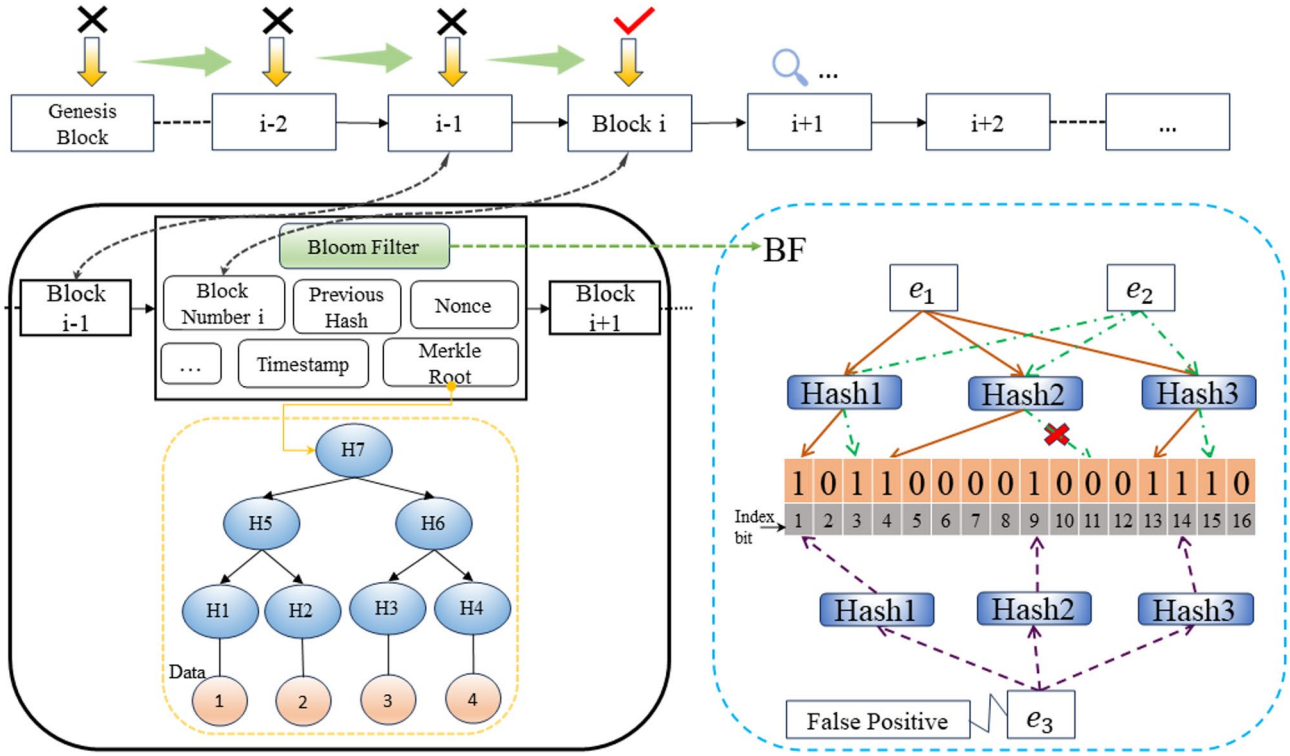


Fig. 1. Bloom filter-based block-by-block lookup process.

length (m) and the number of hash functions (k). To construct a Bloom filter containing n data entries, the selection of filter length m and hash functions k is determined by Formula([1]):

$$k = \frac{m}{n} ln2, m = -\frac{nlnp}{ln2} \quad (1)$$

Here, the false-positive rate p of Bloom filters corresponding to different k and m/n values can be referenced through the Bloom filter parameter table[22].

However, this block-header-embedded Bloom filter query approach still requires sequential block-by-block Bloom filter comparisons, resulting in suboptimal query efficiency. To address this limitation, this paper proposes an aggregated Bloom filter mechanism that eliminates irrelevant blocks through batch verification, thereby avoiding exhaustive block-by-block searches and significantly improving query efficiency.

Interblock search process

Blockchain data are stored in the leaf nodes of the Merkle tree within the block body. The leaf nodes directly contain either raw data entries or their hash digests, whereas nonleaf nodes are constructed by recursively hashing the concatenated values of their child nodes. Through this hierarchical hashing process, the Merkle tree ultimately produces a root hash that ensures data integrity and consistency.

Intrablock queries aim to increase the efficiency of Merkle tree lookups. As data are concentrated in leaf nodes, their structure can be optimized. The improved structure shown in Fig. 2 demonstrates that Merkle trees can be extended into more efficient structures, such as Merkle-B and Merkle-B + trees. Specifically, the Merkle-B tree increases the query speed by adding indices to nonleaf nodes, whereas the Merkle-B + tree further improves the query performance by incorporating linked list structures in leaf nodes on the basis of the B-tree framework.

However, medical data queries exhibit distinct distribution characteristics of 'hotspot data' (frequently accessed data) and 'low-frequency data'. The data heat level in Fig. 2 represents the query frequency, where higher heat values correspond to greater access counts. Taking the query of Data 4 in Fig. 2 as an example, Merkle, Merkle-B, and Merkle-B + trees lack frequency-based optimizations, resulting in equivalent query times for high- and low-frequency data. Specifically, Merkle trees rely on sequential hash-layer verification, whereas Merkle-B and Merkle-B + trees incorporate index structures; however, their query path lengths remain unchanged irrespective of the data access frequency. Moreover, as the data volume increases, the access efficiency of hotspot data gradually decreases due to increasing tree depth. To address this, we propose a weight prediction model for medical data prioritization and introduce the MH tree structure, which places high-weight data closer to the root node. This design enhances hotspot data retrieval efficiency while mitigating the adverse effects of tree depth on query performance.

Sorted Merkle tree and its existing issues

In lightweight blockchain architectures, the block header retains only the key information, and the full node retains the complete block data. The query process is illustrated in Fig. 3. A light node must verify the query result returned by the full node. When a query result exists, it can be verified using the simple payment verification (SPV) mechanism[10], whose workflow is detailed in the upper-right part of Fig. 3. The SPV mechanism verifies the authenticity of the result by comparing the Merkle root computed from the verification path with the hash root stored in the light node. For example, to verify the existence of element 4, its hash value H2 must be obtained along with hash values H1 and H6. The verification process is as follows: H1 and H2 are combined to compute H5, and H5 and H6 are then hashed to derive H7. If the computed root matches the hash root stored in the light node, the existence and integrity of element 4 are confirmed. These sibling nodes collectively form the Merkle verification path for root reconstruction.

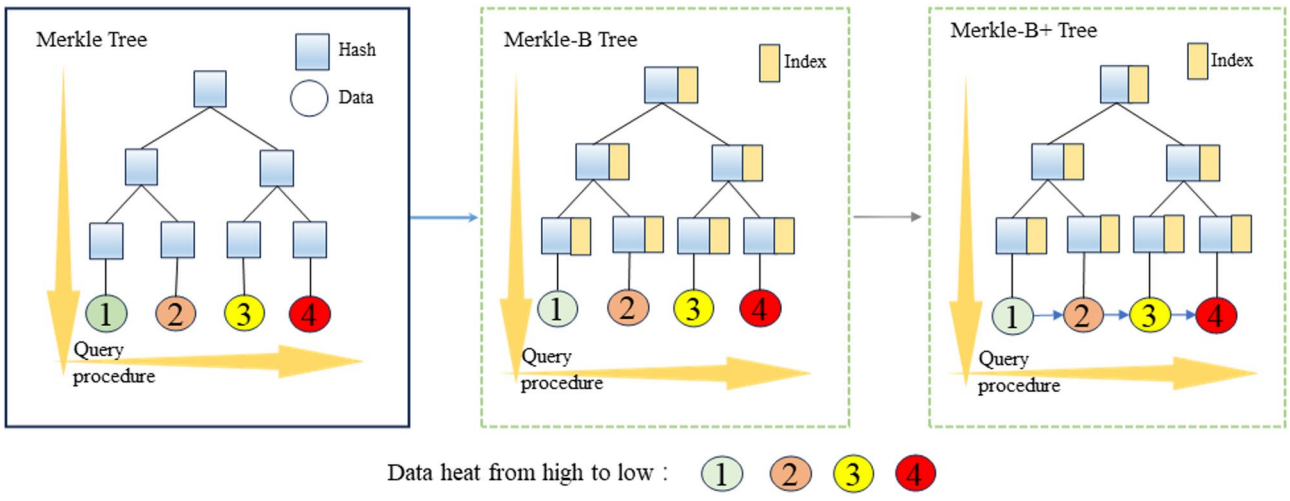


Fig. 2. Query performance comparison across different tree structures.

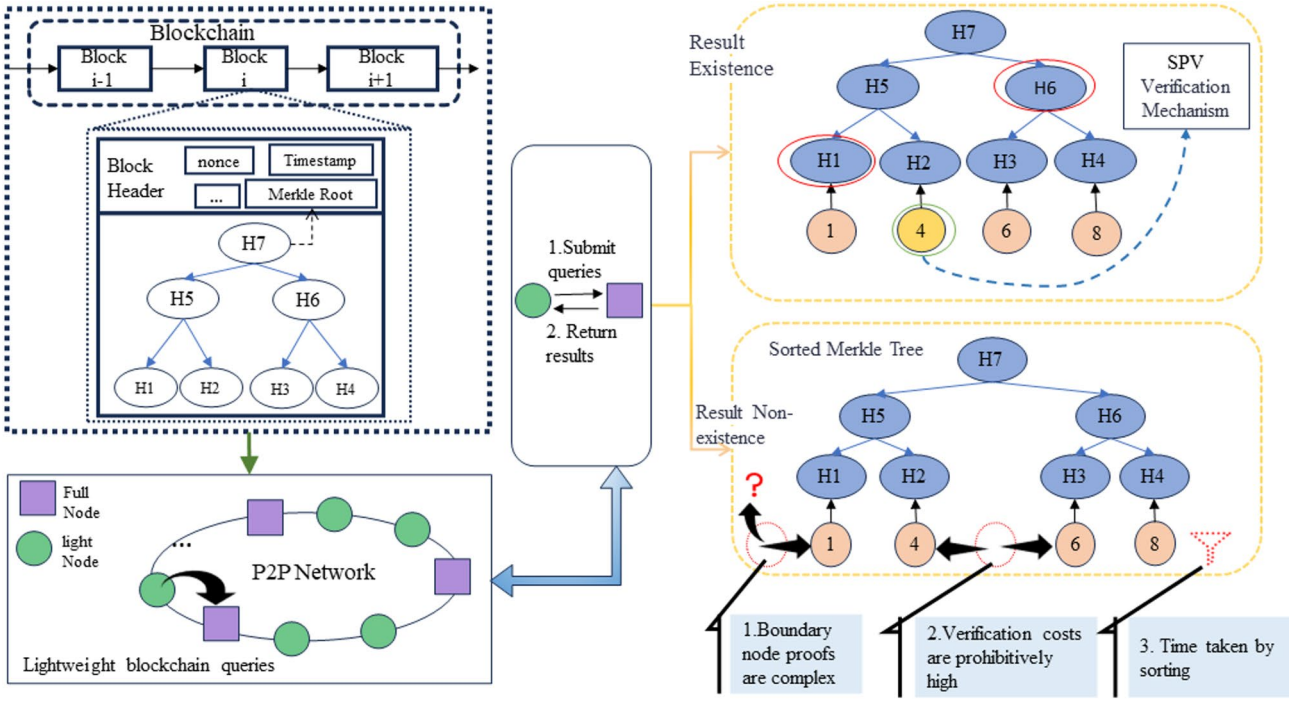


Fig. 3. Lightweight blockchain query process.

When the query results indicate nonexistence, assuming fully trusted nodes, the current solutions primarily employ sorted Merkle trees[10] to validate the accuracy of the results. As depicted in the lower-right section of Fig. 3, sorted Merkle trees arrange all the elements in ascending order within the tree structure, with smaller elements positioned leftward and larger elements rightward. To prove that an element is absent, the sorted Merkle tree must return the Merkle verification paths of its closest predecessor and successor nodes. Since these adjacent elements are verifiably present in the tree, their verification paths can be generated to obtain the Merkle root. However, the sorted Merkle tree's ordered structure has the following limitations:

1. High complexity in verifying boundary nodes. When queried data reside at structural boundaries, auxiliary proof mechanisms are needed to demonstrate nonexistence.
2. A larger proof size and higher verification overhead. Nonexistence proofs necessitate that verification paths be returned for both adjacent nodes, doubling the verification cost compared with single-path proofs for existence verification.
3. Latency in element ordering and potential security risks. The construction of sorted Merkle trees requires data sorting, which introduces significant preprocessing time. Furthermore, the ordered structure may expose data sequence information through order-sensitive proofs.

However, medical data are characterized by large-scale data volumes and high privacy sensitivity. To address the limitations in nonexistence verification mechanisms of sorted Merkle trees, this paper proposes a reputation-based multinode collaborative verification framework. This approach reduces sorting-induced overhead while mitigating information exposure risks inherent to ordered structures.

Weight prediction-based lightweight medical blockchain storage architecture

System architecture

The lightweight medical blockchain storage framework based on weight prediction is architecturally depicted in Fig. 4. The system as a whole consists of three parts: data collection and privacy processing, weight prediction and on-chain storage, and efficient on-chain query for light nodes. Aiming at the problems of low query efficiency and high verification cost faced by directly using traditional Bloom filters in medical blockchain, this paper designs an aggregated Bloom filter(ABF) structure to improve the query efficiency between blocks. By combining the weight prediction model and the MH tree structure, the storage location of data in the Merkle tree is optimized to increase the query speed within blocks. To address the potential false positive issue of Bloom filters, the system further introduces a multi-node collaborative verification mechanism. By collecting and comparing independent verification results from multiple trusted nodes, it ensures the accuracy and credibility of the query results. The specific implementation process of the plan is as follows:

Data collection and privacy processing: As shown in the top layer of Fig. 4, the data management agency is responsible for collecting medical data generated from various hospitals, which includes basic patient information, diagnosis and treatment records, imaging data, test results, etc. To protect patient privacy, the

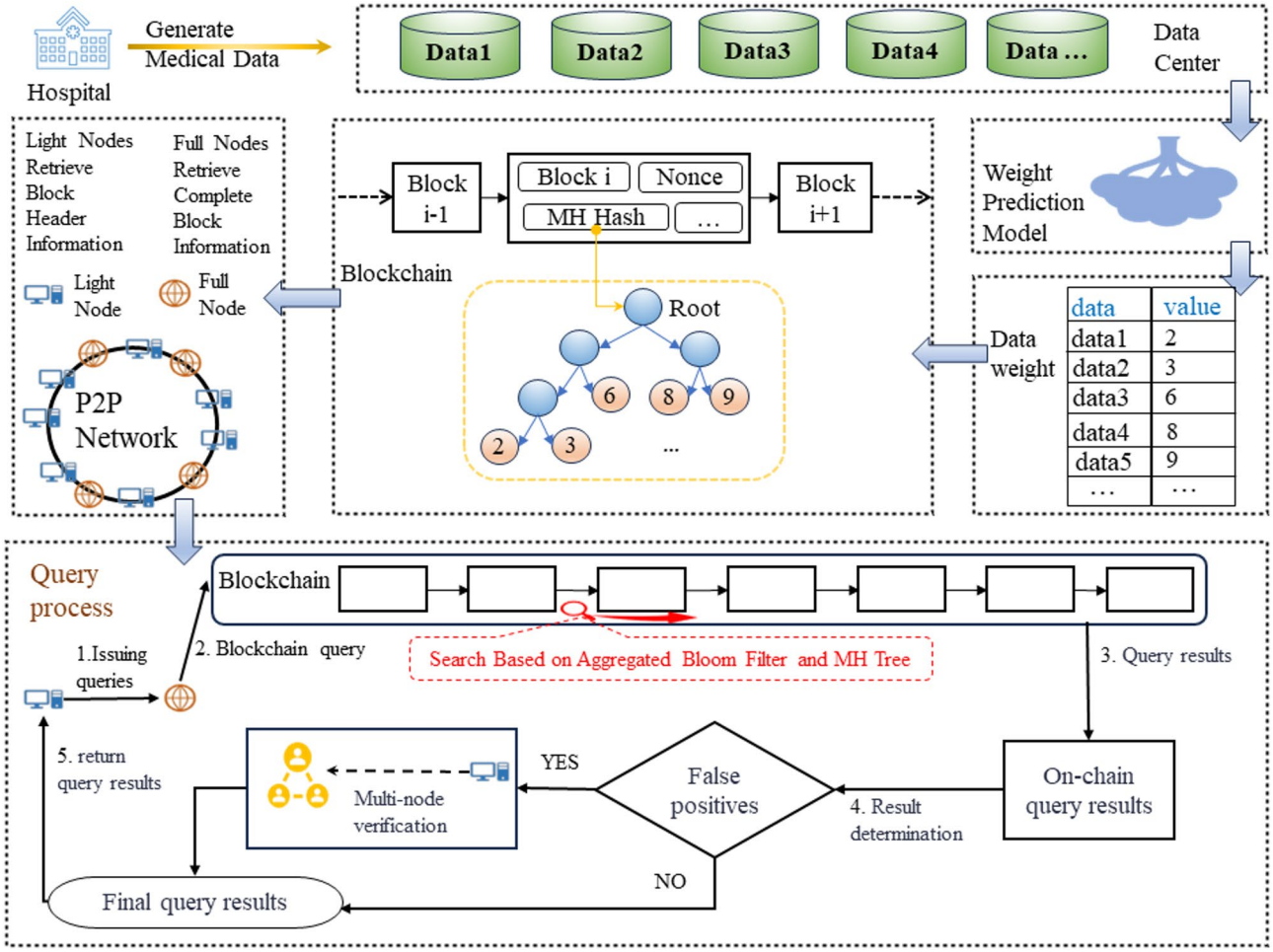


Fig. 4. Weight prediction-based lightweight medical blockchain storage framework.

data center categorizes the collected medical data and applies encryption or data anonymization to personally identifiable and sensitive health information.

Weight prediction and data storage: As shown in the middle layer of Fig. 4, medical data undergo weight prediction before being stored on-chain because of the varying 'values' of different data types. According to the weight prediction results, data are hierarchically organized in the block's Merkle tree, with higher-weight data placed closer to the root node to enhance retrieval efficiency. This medical blockchain adopts a distributed architecture composed of full nodes and light nodes. Full nodes store the complete blockchain data, maintaining network security and integrity despite partial node failures. In contrast, light nodes retain only critical information, such as block headers, reducing storage requirements while improving overall network efficiency.

Query process: As shown in the bottom layer of Fig. 4, During the data query process, the light node acts as a query agent, forwarding the user's query request to the full node to obtain the target data. The efficient search of on-chain data is achieved through the collaboration of aggregated Bloom filters and the MH tree structure: The system first uses ABF to quickly locate the possible block where the target data may be, and then completes the search within the block through the MH tree structure. In the event of the inherent false positive situation of the Bloom filter, the system will automatically trigger a multi-node collaborative verification mechanism. By comparing the verification results of multiple independent reputation nodes, it can eliminate misjudgments and ensure the accuracy of the query results. The complete implementation method of the query process will be elaborated in detail in Sect. 4.

Due to the high sensitivity of medical data, different users have varying access requirements and permissions for the data. Medical institutions usually need to comprehensively search all medical data in disease research or patient treatment, while patients can only view their own medical records, and third-party institutions can only access the data of specific personnel related to them. Therefore, when a proxy query request is initiated to a light node, the light node will return the data that the object can view based on the permission of the object to be accessed, thereby maintaining the security of the system.

Medical data weight prediction

The extreme gradient boosting (XGBoost) algorithm[23] demonstrates superior accuracy and efficiency in predictive tasks, making it the preferred method for training our medical data weight prediction model. The

subsequent data are assigned differentiated weights through this trained model. XGBoost's core principle is to strategically combine multiple weak learners (decision trees) into a high-performance ensemble model, significantly enhancing the prediction accuracy. Furthermore, regularization techniques mitigate overfitting risks, ensuring that the model can be generalized effectively to unseen data. The objective function constructed from each subtree is formulated as follows:

$$f_{OBJ}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{k} \Omega\left(f_k\right) \tag{2}$$

Where $l(y_i, \hat{y}_i^{(t)})$ denotes the loss function, which quantifies the model's fitting performance; $\Omega(f_k)$ represents the regularization term, introduced to mitigate overfitting; y_i denotes the ground truth, and $\hat{y}_i^{(t)}$ represents the prediction of the t-th decision tree. In this study, key medical data features are extracted, and XGBoost is leveraged to construct a data weight prediction model.

Since a single medical data record contains numerous attributes, this paper selects only three features for model training: basic information (including age, gender, etc.), health characteristics (including height, weight, behavioral habits, etc.), and medical information (including treatment cycle, medication usage, diagnostic data, etc.). These features can be used to assess the value of medical data accurately in most cases. The XGBoost-based medical data weight prediction model is shown in Fig. 5. The model determines the weight of data by extracting these three features (basic information, health characteristics, and medical information) from raw data. Considering the inconsistency and diversity of medical data types, we extract the following factors from these three features as weight determinants: age, body mass index (BMI), treatment cycle, behavioral habits, sex, admission status, and diagnostic data.

The three categories of features in the medical dataset include both numerical and categorical variables. To enhance model performance and reproducibility, different transformation strategies were employed during the data preprocessing stage to achieve a unified representation of features. Categorical variables, including gender, behavioral habits, admission status, and diagnostic information, were uniformly encoded using label encoding. This method assigns a unique integer label to each category, thereby converting original discrete variables into numerical features that are compatible with the model. Numerical variables, such as age, BMI, and treatment duration, were normalized using the Z-score standardization method. This approach transforms each feature into a standard normal distribution with a mean of 0 and a standard deviation of 1 based on the statistical properties (mean μ and standard deviation σ) of the training data, thereby achieving feature standardization. The data processing steps are shown in the upper part of Fig. 5.

During the model training stage, XGBoost trains multiple weak learners (decision trees) in an iterative manner to progressively optimize the predictive performance. Initially, the first weak learner generates predictions on the basis of input features and computes residual errors. Each subsequent weak learner is trained using the residuals from the previous iteration as the target variable, gradually reducing the number of prediction errors. Finally, a strong learner is formed by aggregating the weighted outputs of all weak learners to predict medical data weights. The model training process is depicted in the lower section of Fig. 5, while Algorithm 1 provides the detailed implementation of the weight prediction model.

To evaluate the predictive capability of the proposed weight prediction model, the coefficient of determination (R^2), root mean square error (*RMSE*), and mean absolute error (*MAE*) are adopted for assessment. The R^2 value reflects the model's goodness-of-fit to the sample data, with values closer to 1 indicating that the model effectively explains the variance of the target variable, demonstrating strong predictive performance; a smaller *RMSE* signifies that the model's predictions are closer to the actual values; and the *MAE* represents the average

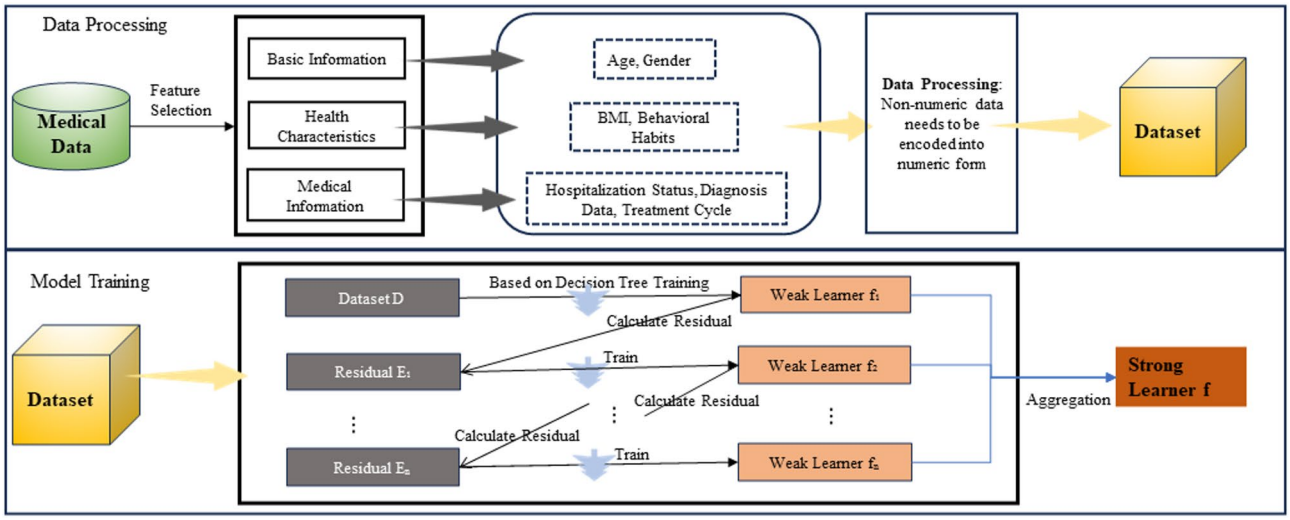


Fig. 5. XGBoost-based medical data weight prediction model.

Input: $Data\ D = (x_i, y_i)$

//Extract features from medical data x_i ,where y_i denotes the true weight of the data.

Output: *Medical data weight prediction model f*

```
features = select_features(D)
D_processed = [ ]
for each feature in features:
    if feature is categorical:
        feature_encoded = LabelEncode(feature)
    else if feature is numerical:
        feature_normalized = (feature − μ) / σ
            /*Z-score Normalization (where μ is the mean of the training set and σ is the standard
    deviation of the training set)*/
end for
D(x_{process}) = D_processed
f(x) = 0
for i = 1 to n :          /*Iterative Gradient Boosting*/
    residual = compute_rsidual(D, f)
    residual = y - f(x)
    f_i(x) = train weak learner(x, residual)
    f = f + α · f_i
end for
return f
```

Algorithm 1 XGBoost-based medical data weight prediction model.

of the absolute differences between the predicted and actual values, reflecting the average magnitude of the prediction errors. The relevant calculation formulas are as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y})^2}{\sum_{i=1}^{n} (y_i - \overline{y_i})^2} \tag{3}$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y})^2} \tag{4}$$

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |(y_i - \hat{y})| \tag{5}$$

where n is the number of samples in the test set; y_i is the actual value of the test sample; $\hat{y}$ is the predicted value of the test sample; and $\overline{y_i}$ is the mean of the actual values in the test set.

Data query methods and nonexistence proofs

Data query based on an aggregated bloom filter and MH tree

Due to the huge scale of medical data, the traditional blockchain structure has a relatively low efficiency in querying each block one by one. For inter-block and intra-block queries, this paper respectively employs aggregated Bloom filters and MH tree structures to enhance query efficiency.

InterBlock search via an aggregated Bloom filter matrix

Introducing a Bloom filter into the block header can avoid intrablock queries for irrelevant blocks. However, this method still requires sequential comparisons across blocks, prompting an enhancement of the Bloom filter-based interblock search mechanism.

Starting from the genesis block, every n consecutive blocks form a segment. Each block's Bloom filter uses a fixed-length bit array of m bits, creating an $n \times m$ bit matrix. Transposing this matrix enables rapid filtering of irrelevant blocks through correlation calculations on its row vectors. Figure 6 illustrates the intersegment search process using the aggregated Bloom filter matrix (ABFM): Blocks i to $i+5$ (with $n=6$ and $m=8$) form a segment. Their Bloom filter bit matrix is transposed. Rows represent vectors composed of all segment blocks' Bloom filter bits, whereas columns correspond to individual blocks' Bloom filters.

By segmenting blocks and constructing the ABFM within segments, irrelevant blocks can be filtered directly, eliminating sequential block-by-block searches. For a given query keyword w, the query process is as follows: The hash value of w is computed and converted into a Bloom filter-related binary code; row vectors are selected where all hash-mapped bit positions are 1; bitwise AND operations are performed on the corresponding row vectors in the ABFM to identify candidate blocks. Since Bloom filters may produce false positives, blocks falsely identified as containing the query result must undergo an intrablock lookup step. If no matching data are found, a nonexistence proof is returned.

Taking Fig. 6 as an example, assume that hashing the query data results in the bit sequence 10,100,001 (with the Bloom filter length set to 8 and the number of feature fields set to 3). Within this block segment, if the first, third, and eighth bits in a block header's Bloom filter are all set to 1, that block may contain the query result. The bit positions set to 1 in 10,100,001 correspond directly to the first, third, and eighth rows of the transposed Bloom filter matrix. Their respective row vectors are 100,111 (Row 1), 111,010 (Row 3), and 101,111 (Row 8). Applying a bitwise AND operation to these three row vectors yields 100,010. The bits set to 1 in the first and fifth positions indicate that Block i and Block $i+4$ (corresponding to positions 1 and 5 in Fig. 6) within the segment may contain the query result. All other blocks in the segment can be immediately disregarded, enabling rapid filtering of irrelevant blocks.

Intrablock search based on the MH tree

To improve query efficiency, incorporates the concept of the Huffman tree into the Merkle tree framework, constructing a hybrid structure termed the Merkle-Huffman Tree (MH Tree). The core methodology is as follows: when data is stored on the blockchain, a weight prediction model assigns a weight to each data item, and data with higher weights are positioned closer to the root node. This approach retains the rapid verification capability of the Merkle tree while enhancing query efficiency by reducing the average access path for high-frequency data. Taking the query for data node a3 in Fig. 7 as an example, the verification path in the conventional Merkle tree structure consists of nodes a4, H_{12}, and H_5 (left side of Fig. 7); whereas in the MH Tree structure, the verification path comprises nodes H_{12} and H_{45} (right side of Fig. 7). The Weighted Path Length (WPL) for both the Merkle tree and the MH Tree structures are presented in formulas (6) and (7), respectively:

$$\text{Merkle}_{WPL} = \sum_{i=1}^{n} w_i \cdot d_i = (3+6+12+15+20) * 3 = 168 \tag{6}$$

$$\text{MH}_{WPL} = \sum_{i=1}^{n} w_i \cdot d_i = (3+6) * 3 + (12+15+20) * 2 = 121 \tag{7}$$

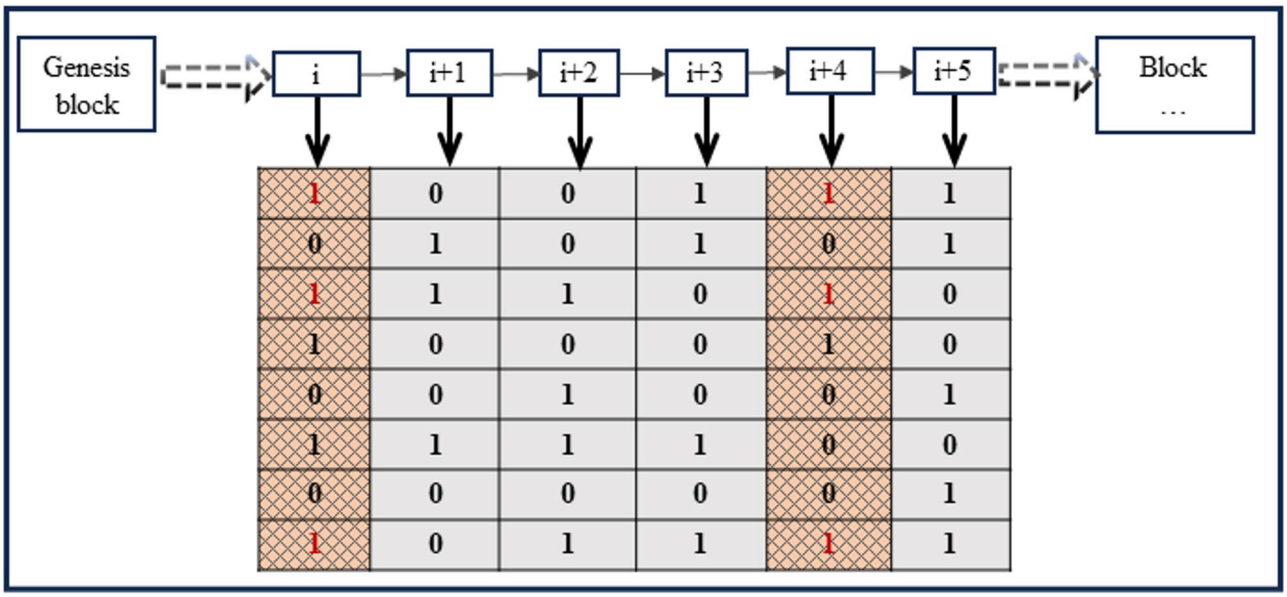


Fig. 6. Intrasegment aggregated-Bloom-filter-based lookup process.

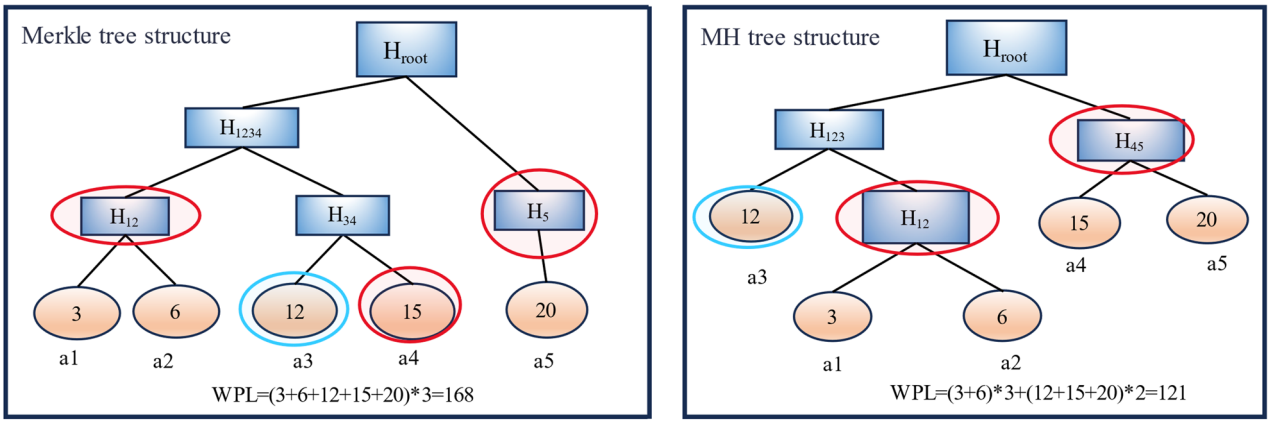


Fig. 7. Query structures of Merkle trees and MH trees.

Input: *block number i*; *target node T*

Output: *result* (*result* = 1 indicates that *T* exists in block *i*, *result* = 0 indicates absence)

```
result = 0
Initialize queue Q ← [(root, [])]
while Q is not empty do:
    (node, current_path) ← Dequeue(Q)
    if node == T then
        if validate_path(current_path) then
            result = 1
            break
        end if
    else
        if node has children then
            for each child c of node do
                sibling_hash ← get_sibling_hash(c)
                updated_path ← current_path + [sibling_hash]
                Enqueue(Q, (c, updated_path))
            end for
        end if
    end if
end while
return result
```

Algorithm 2 MH Tree-based Intrablock Search Algorithm.

Here, n represents the number of nodes, w_i denotes the weight of the data item, and d_i indicates the height of the leaf node within the tree. The WPL value is 121 for the MH tree but 168 for the Merkle tree. The lower WPL of the MH tree results in shorter average access paths during queries, reducing the traversal distance of intermediate nodes. This optimization simultaneously improves the query efficiency and enhances the data verification performance.

The intrablock search process based on the MH tree is detailed in Algorithm 2. Leveraging the MH tree's characteristics, a breadth-first search (BFS) algorithm is employed for intrablock queries. During the search, the queue stores both the current node's information and relevant sibling node data to facilitate Merkle verification path generation.

In the MH-Tree structure, data weights are determined and fixed by a prediction model during block generation. However, in practical application scenarios, low-weight data may experience sudden spikes in access demand. The lack of a dynamic adjustment mechanism could compromise overall query performance. To address this, a sliding window is introduced in the block header to monitor in real time the access frequency of data within the block. Each time a new query is added to the window, the earliest query record is removed. The popularity factor h_i for a data item i is calculated based on its access count f_i within the window W, as follows:$\mathrm{h_i} = f_i / W$. The comprehensive weight of the data is jointly determined by the predicted weight and the dynamic popularity factor:

$$w_i^{final} = \lambda \cdot w_i^{pred} + (1 - \lambda) h_i \tag{8}$$

Here, w_i^{pred} denotes the static weight computed by the prediction model during block generation, and the parameter $\lambda \in [0, 1]$ is a balancing coefficient that adjusts the influence of static prediction and dynamic access patterns. By dynamically adjusting the comprehensive weight of low-weight data, adaptive optimization for sudden access patterns can be achieved without modifying the on-chain MH-Tree structure.

Node reputation evaluation mechanism

In the proposed scheme, if the aggregated Bloom filter returns no false positives during a query, its result directly proves that data do not exist. When false positives occur, a 'data nonexistence proof' must be generated. To address the challenges posed by Bloom filter false positives, this paper introduces a reputation-based multinode collaborative verification mechanism. By leveraging collaborative verification from multiple nodes to prove the nonexistence of query results, this mechanism compensates for the Bloom filter's limitations in false-positive scenarios, thereby enhancing the security and reliability of the overall verification scheme.

Node reputation model

In the decentralized network environment, a node reputation evaluation mechanism is introduced because the trustworthiness of a single node cannot be fully guaranteed. When a lightweight node selects a full node to send a query request, the reputation score of the full node is used as a core indicator to evaluate its trustworthiness. The higher the reputation score is, the greater its reliability, and the more likely it is that a lightweight node will send a query to such a node. The reputation of a node is determined by four factors: the query accuracy, response time, query participation rate, and reputation decay mechanism. In practice, the query frequency of health care blockchains is usually very high. Broadcasting reputation updates to all nodes after each query would incur considerable communication overhead. To solve this problem, the proposed scheme uses a periodic adjustment strategy to calculate the reputation. After completing a query cycle, the reputation scores of all nodes are updated in bulk. The node reputation calculation is formalized in formula (9):

$$\mathrm{R_{new}} = \mathrm{R_{pre}} + \mathrm{R_{query}} + \mathrm{D} + \mathrm{R_{res}} + \mathrm{R_p} \tag{9}$$

In this context, $\mathrm{R_{pre}}$ denotes the node's reputation after completing the previous query cycle, $\mathrm{R_{query}}$ represents the reputation value associated with query operations, D represents the reputation decay mechanism, $\mathrm{R_{res}}$ corresponds to response time performance, and $\mathrm{R_p}$ indicates the number of node recommendations.

In formula (9), $\mathrm{R_{query}}$ represents the query-specific reputation value. When a full node provides valid query results, its reputation increases; conversely, if it acts maliciously by supplying false results, the reputation decreases. To prevent malicious nodes from rapidly restoring their reputation after intentional misconduct, the reputation adjustment for query operations follows a nonlinear pattern. The query-related reputation calculation is defined in formula (10):

$$R_{query} = \alpha \cdot \ln(1 + s_c) - \beta \cdot e^{f_c} \tag{10}$$

Here, α and β denote the correlation coefficients, whereas s_c and e^{f_c} represent the numbers of successes and failures for queries within a single query cycle, respectively.

In formula (9), D represents the reputation decay mechanism. To prevent unbounded growth of node reputation scores, a natural decay mechanism is introduced, thereby avoiding scenarios where nodes maintain excessively high reputation scores indefinitely after providing correct query results over time. If a node's reputation consistently remains the highest, lightweight nodes will increasingly prioritize sending queries to this full node, which risks transforming it into an 'oligopoly' node. The reputation decay calculation is formalized in formula (11):

$$D = \gamma \cdot e^{-kt} \tag{11}$$

Here, γ and k denote correlation coefficients, and t represents a time period.

In formula (9), $\mathrm{R_{res}}$ corresponds to the response time metric, which reflects the node's bandwidth resources and related network performance indicators. The calculation of $\mathrm{R_{res}}$ is defined in formula (12):

$$R_{res} = \delta \cdot (q_t - r_t) \tag{12}$$

Here, δ denotes the correlation coefficient, q_t represents the node's response time for the current query, and r_t indicates the minimum response time threshold.

In formula (9), R_p corresponds to the number of recommendations, which quantifies how frequently the node participates in queries within a cycle. This metric reflects the node's proactiveness in query operations. The calculation of R_p is defined in formula (13):

$$R_p = \epsilon \cdot \ln(1 + x) \tag{13}$$

Here, ϵ denotes the correlation coefficient, and x represents the participation count.

By integrating formulas (10)–(13), the node reputation within a query cycle is calculated as shown in formula (14):

$$R_{new} = R_{pre} + \alpha \cdot \ln(1 + s_c) - \beta \cdot e^{f_c} - \gamma \cdot e^{-kt} + \delta \cdot (q_t - r_t) + \epsilon \cdot \ln(1 + x) \tag{14}$$

Here, the coefficients α and β are query related and have the greatest impact on node reputation scores, with assigned values of 0.1 and 0.8, respectively; γ, δ, and ϵ are set to 0.1; and k is 0.5. These coefficients are dynamically adjustable to accommodate varying query requirements under different conditions.

Reputation-based multinode nonexistence proof

When a Bloom filter generates false positives, leading to misjudgments, the verification result from a single node may fail to reflect the actual scenario. To address this shortcoming, this paper proposes a multinode collaborative verification mechanism. Lightweight nodes broadcast query requests to multiple full nodes and synthesize verification results from these nodes to reach consensus, thereby enhancing the accuracy and reliability of verification.

Node reputation scores are defined within the range of 0–100. For nodes with reputation scores below 50, a tiered penalty mechanism is implemented. For light penalties, when a full node's reputation falls below the lower threshold (50), the system reduces its query response priority. For moderate penalties, if the node's reputation persistently decreases below 40, the system restricts its participation frequency in query operations. For severe penalties, when a full node's reputation score decrease to below 30, the system permanently removes the node and broadcasts its malicious behavior to the entire network.

The multinode collaborative verification mechanism is illustrated in Fig. 8. A lightweight node randomly selects a full node with a reputation score above 60 to initiate a query, and the selected full node must return the final query result along with supporting proof information. If the query result indicates nonexistence, a multinode verification process is triggered. The verification process proceeds as follows: First, the number of verification nodes is determined on the basis of the reputation score of the initial full node that provided the query result, with all verification nodes required to have reputation scores exceeding 70. If the initial node's reputation score is 95 or higher, 3 verification nodes are selected. For every 5-point decrease in reputation below 95, one additional verification node is added. The logic for determining the number of verification nodes is formalized in formula (15):

$$N = \begin{cases} 3, R \geqslant 95 \\ 3 + \left\lceil \frac{95-R}{5} \right\rceil, 70 \leqslant R < 95 \end{cases} \tag{15}$$

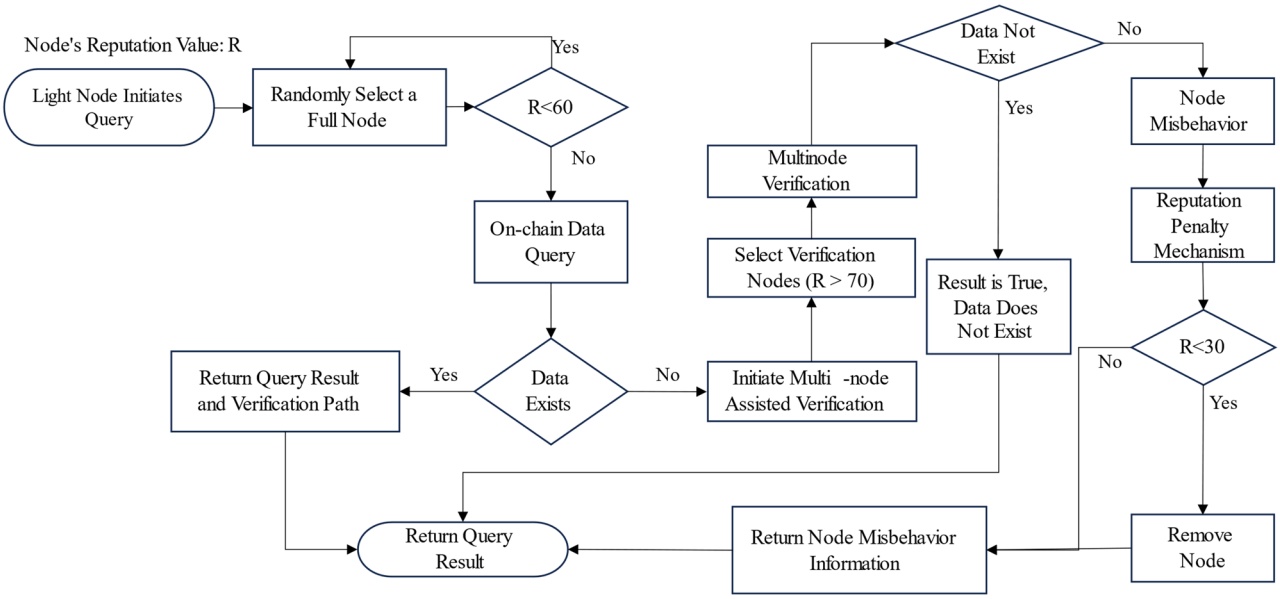


Fig. 8. Multinode collaborative verification flowchart.

Here, R denotes the node's reputation score, N represents the number of verification nodes, and $\lceil x \rceil$ indicates the ceiling function. Verification nodes must have a reputation score of 70 or higher. The verification nodes then perform consensus validation on the query result. If the consensus confirms the data's existence, full nodes that falsely reported nonexistence are penalized by reducing their reputation scores. Nodes with reputation scores below a critical threshold are permanently removed from the network. Finally, the system returns the definitive query result, terminating the verification process.

The multinode collaborative verification mechanism not only ensures the authenticity and consistency of the query results but also mitigates the risks inherent to single-node verification. In the context of medical blockchain applications, where most data are historically authentic, the probability of nonexistent data is inherently low. Through optimized parameter configuration, the Bloom filter's false-positive rate can be confined within an acceptable range, ensuring that the resource overhead of multinode verification remains manageable. Furthermore, the proposed penalty mechanism enhances system security by removing nodes with critically low reputation scores, thereby reducing the likelihood of malicious behavior.

Feasibility analysis of the proposed scheme

Theoretical analysis

This section provides a theoretical analysis of the proposed scheme in terms of query efficiency and security, demonstrating its ability to enable efficient and secure medical data queries.

Query efficiency of the aggregated bloom filter and MH tree

Existing query schemes typically require sequential traversal of blocks with a query time complexity of $O(log\ n)$. When handling large-scale datasets, this method leads to extended query latency and diminished system efficiency. By implementing the aggregated Bloom filter, relevant blocks can be directly identified without iterative searches, thereby reducing the query time complexity to below $O(log\ n)$.

Within the block structure, the leaf nodes of the Merkle tree store specific data, and its query efficiency depends on the depth of the tree. As the volume of data within a block increases, the required query depth increases proportionally, thereby reducing the query performance. For a block containing n nodes, the Merkle tree requires traversal of all nodes during queries, resulting in an average query time of $n/2$ and a time complexity of $O(n)$. In contrast, the MH tree prioritizes high-weight data (data with greater significance) by storing them closer to the root node, enabling faster discovery of high-value data during queries. Consequently, the MH tree achieves an average query time below $n/2$, although its time complexity remains $O(n)$.

After the data weights are predicted via a weight prediction model, each data item is assigned a weight denoted w_i $(i = 1,2,3\ldots,n)$. The average path length of the MH tree is calculated as shown in formula (16):

$$H\left(mh\right) = \sum_{i=1}^{n} w_i \cdot d_i \tag{16}$$

Here, $w_i$represents the weight of a data item, defined as its query probability, and d_i denotes the depth of data item i in the MH tree. To minimize the total query cost, data items with higher weights are assigned smaller depths, thereby optimizing path traversal costs.

In Merkle trees, the node height distribution is uniform. For any leaf node, its path length equals the number of edges from the node to the root, which is equivalent to its depth. The average path length of a Merkle tree is calculated as shown in formula (17):

$$H(m) = \sum_{i=1}^{n} w_i \cdot h_i \tag{17}$$

Here, w_i denotes the weight of the data item, and h_i represents the height of the leaf node in the tree.

Compared with the Merkle tree approach, the MH tree reduces the path length for high-frequency data through weight-based optimization, thereby ensuring an optimal average path length. Since not all the elements in the MH tree reside at the lowest level, this design guarantees that the average path length of the MH tree is consistently less than or equal to that of the Merkle tree. Consequently, the inequality $d_i \geqslant h_i$ holds. From formula (16) and sormula (17), we derive the following:

$$H\left(m\right) - H\left(mh\right) \geqslant 0 \tag{18}$$

This proves that the inequality $H\left(mh\right) \leqslant H\left(m\right)$ holds, demonstrating that the MH tree achieves superior query performance over the Merkle tree in high-frequency data access scenarios. For infrequently accessed data, which exhibit lower query probabilities, these entries typically reside at relatively deeper levels within the MH-tree structure, thereby incurring longer query paths. Nevertheless, as such data demonstrate significantly reduced access frequencies in practical operational systems, their impact on overall query performance remains operationally manageable.

Security analysis

The proposed framework adopts a "light-node query, full-node maintenance" architecture, which is further reinforced by reputation-based request scheduling and hierarchical access control. This design not only preserves query efficiency but also enhances the overall security and robustness of the system.

Malicious node attacks: When multinode collaborative verification is adopted and malicious nodes attempt to return incorrect verification results, their probability of a successful attack depends on the proportion of malicious nodes and the size of the verification node set. Let p denote the proportion of malicious nodes and $|\nu|$ represent the verification set size. The probability of tampering with the final verification result is calculated as shown in formula (19)

$$P_{attack} = p^{|\nu|} \tag{19}$$

To mitigate attack risk, the verification set size p can be adjusted. For example, when $p = 0.3$ (the malicious node proportion is 30%) and $|\nu| = 6$,

$$P_{attack} = p^{|\nu|} = 0.3^6 = 0.000729 \tag{20}$$

This indicates that the probability of a successful attack is only 0.07%, demonstrating the system's ability to effectively resist malicious node attacks.

False-Positive attacks: Bloom filters may generate false positives, causing certain query results to be erroneously flagged as existing. The false-positive probability p_f is calculated as shown in formula (21).

$$P_f = \left(1 - e^{-\frac{k \cdot n}{m}}\right)^k \tag{21}$$

Here, k denotes the number of hash functions, m represents the size of the Bloom filter, and n is the number of data items. By adjusting parameters k and m, the false-positive probability can be constrained within an acceptable range. For example, when $k = 3$, n=1000 and $m = 10^4$, the calculated false-positive probability is $P_f \approx 0.008$. Therefore, even if an attacker launches large-scale malicious queries, the proportion of false positives that can be triggered remains strictly bounded. Moreover, if the system detects that a particular source generates frequent false positives within a short period, it will apply rate-limiting and penalization strategies based on the reputation model, thereby preventing sustained resource exhaustion.

Query heat-manipulation attack: Within the sliding-window-based weight update mechanism, an attacker may attempt to increase the short-term access frequency of a low-weight record by repeatedly querying it, thereby raising the heat factor and reducing query latency. However, since the MH-tree structure inside each block is fixed at the time of block generation, such operations can only affect local query path optimization but cannot alter the on-chain storage structure or global verification logic. Assuming a sliding window size of W, if the attacker issues q valid requests within the window, the maximum increase in the composite weight is given by.

$$\Delta \mathrm{w_i} = (1 - \lambda) \cdot \frac{q}{W} \tag{22}$$

When a rate-limiting threshold $Q_{\max}$ is enforced on the number of queries from a single source within a time window, this upper bound is further constrained to$(1 - \lambda)Q_{\max}/W$. For example, when W=1000,$\lambda = 0.6$, and $Q_{\max} = 50$, the maximum weight increase is only 0.02, which is negligible. More importantly, the selection of verification nodes is entirely determined by their reputation values, which are independent of query heat and dynamically adjusted through a penalty mechanism. Therefore, even if an attacker artificially inflates query frequency to boost data heat, it does not increase their probability of being chosen as a verification node.

Security of nonexistence proofs: The MH tree integrated with Bloom filters enables efficient nonexistence verification. When no false-positive occurs in the Bloom filter, the query result is directly returned; if a false-positive is triggered, the multinode collaborative verification mechanism ensures result credibility. The consensus outcome of multinode verification follows weighted voting, with the validation process defined in formula (23).

$$R^* = \arg\max_{r \in \{0,1\}} \sum_{N_i \in V} \mathbb{I}(R_i = r) \cdot C_i \tag{23}$$

Here, R^* represents the final result (1 indicates data existence, 0 indicates nonexistence); $R_i$denotes the verification result from node N_i; C_i is the credibility value of node N_i; and $\mathbb{I}(R_i = r)$ is the indicator function, which equals 1 if $R_i = r$ and 0 otherwise. If a node repeatedly returns incorrect results, its credibility value will drop below the threshold ($C_{\min}$), and it will be removed from the system. Let C_{init} be the initial credibility value of a node and let λ be its malicious behavior frequency. The time until its removal is calculated as follows:

$$T_{ban} = \frac{C_{init} - C_{min}}{\lambda \cdot \beta} \tag{24}$$

Here, β denotes the penalty coefficient for query failure. By configuring a lower $C_{\min}$ and a higher β, malicious nodes can be swiftly identified and eliminated.

Experimental validation

To verify the effectiveness and superiority of the proposed scheme, the experimental environment was configured with an Intel(R) Core(TM) i5-8250U CPU @ 1.60 GHz processor, 12GB of memory, and the Windows 10 operating system. The software environment included PyCharm 2023.1 and Matlab R2019a, and Ubuntu 18.04

LTS instances were created via virtual machines to simulate the deployment of the distributed blockchain network. The experimental dataset was obtained from an open-source medical dataset on the Kaggle platform (approximately 55,000 records). An XGBoost algorithm was used to train the weight prediction model, and each record was assigned a weight and stored in the blockchain. The experimental network consisted of 10 full nodes and 50 light nodes, connected via a random graph topology to simulate peer-to-peer communication. Full nodes were responsible for data storage and verification, while lightweight nodes only performed query operations. In addition, about 5% of the nodes were configured as malicious to test the effectiveness of the collaborative verification mechanism. The experimental parameters were set as a block size of 1 MB and an average block interval of 10 s, with query requests generated according to a Poisson distribution to approximate the access patterns of real-world medical scenarios.

Medical data weight prediction experiment

The experiment utilized the first 10,000 entries from the dataset for model training, which were divided into a training set (8,000 entries) and a validation set (2,000 entries). We first calculated the ground-truth weights of the training data using hierarchical comprehensive evaluation and then compared them with the predicted weights to evaluate the model's performance. Figure 9 shows the discrepancies between the true weights and the predicted weights on the validation set, where the x-axis represents the validation set entries and the y-axis indicates the data weights. Through R^2 and *RMSE* calculations, the proposed model achieved 96.422% prediction accuracy on the validation set. The results confirm strong alignment between the predicted and true data weights, demonstrating high model precision for effective weight prediction.

Additionally, the experiment evaluated the model's learning capability and generalization performance through learning curves and errors on both the validation and test sets. Figure 10(a) shows the learning curve for the training set, illustrating the model's performance with varying training sample sizes. When the training set size is 0, the model cannot be trained; hence, no error values exist. The experiment started calculations from a 10% sample ratio to ensure reasonable training progression. In Fig. 10(a), the blue solid line represents the training error, indicating the trend of the error on the training set as the sample size increases. The red dashed line denotes the validation error, reflecting the model's performance on the validation set across training set sizes. As the number of samples increases, the model's errors on both the training set and the validation set decrease, indicating that when the number of training set samples is small, the model has not yet fully learned the data features. When the number of training samples is about 2,200, the error tends to stabilize, indicating that the model has sufficient learning ability under this data scale, and further increasing the number of samples will have limited impact on improving model performance. After the training error stabilized, it stabilized at about 2.0×10^{-5}, which is a small error, indicating that the model has a good fit to the training set data. After the validation error stabilized, it stabilized at about 2.3×10^{-5}, and the difference between the validation error and the training error was small, indicating that the model performed equally well on unseen data and had good generalization ability. In addition, there was no obvious separation between the two curves, which also showed that the model did not overfit.

Figure 10(b) displays the *RMSE* and *MAE* for both the validation and training sets. The *RMSE* values for the training and validation sets are 0.00438 and 0.00455, respectively, whereas the *MAE* values are 0.00356 and 0.0036, respectively. Overall, the gap between the training error and the validation error is small, which further verifies the consistency and robustness of the model. The lower *MAE* value indicates that most of the model's predictions are closer to the true values and have high practical value in practical applications.

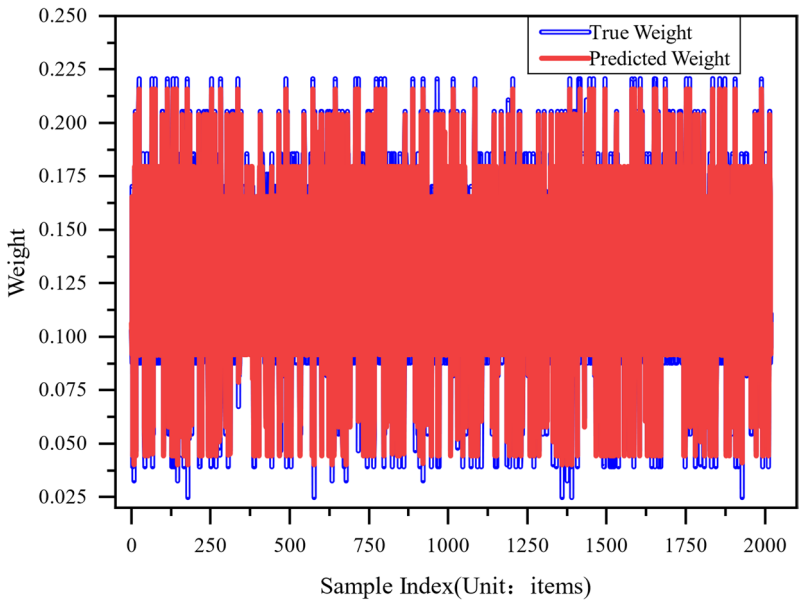
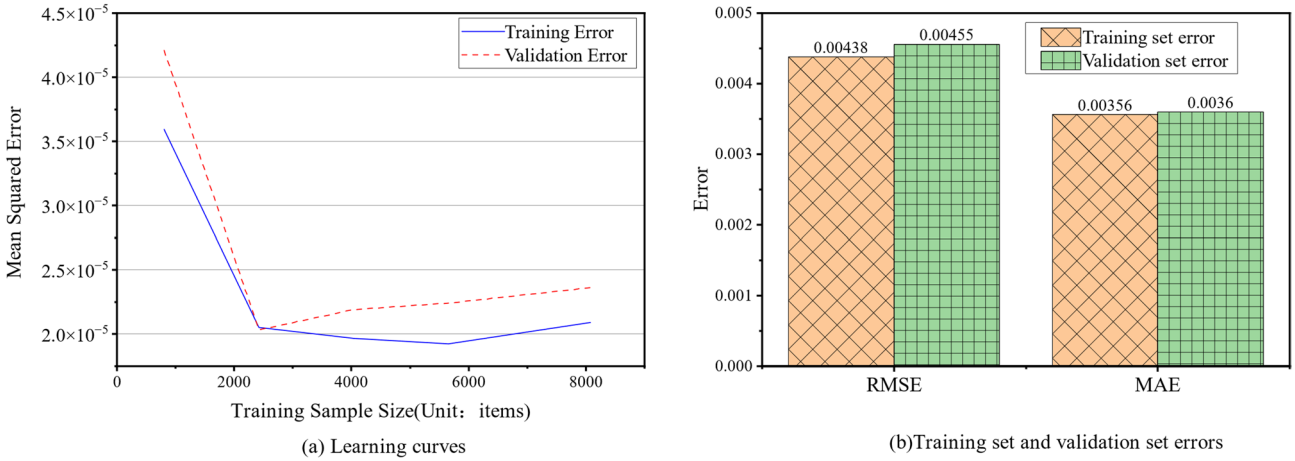


Fig. 9. Prediction accuracy.



(a) Learning curves

(b)Training set and validation set errors

Fig. 10. Learning curves and Training set and validation set errors.

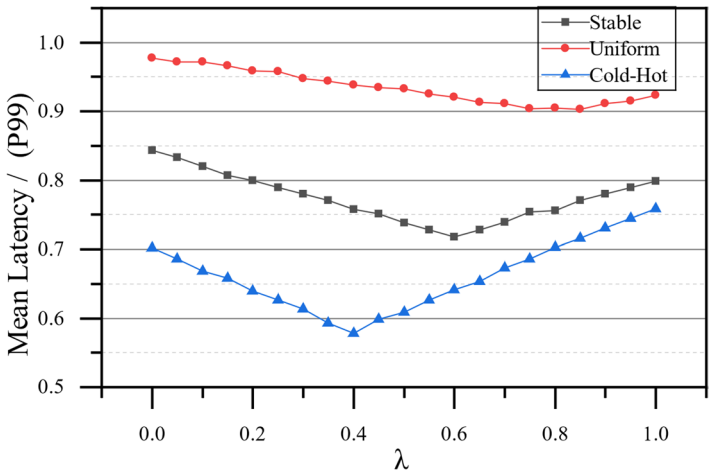


Fig. 11. Selection of parameter λ.

From the prediction accuracy, learning curve trend, and comparison of errors between training and validation sets, it can be seen that the proposed model has good learning ability and generalization performance on this dataset, and can accurately and efficiently complete the task of medical data weight prediction, providing a solid foundation for subsequent data management and blockchain query.

On-chain data query experiment

In the sliding-window weight update mechanism of the MH tree, the parameter $\lambda \in [0, 1]$ is employed to balance the contributions of the static prediction weight and the dynamic access heat factor, and its value directly influences query performance. To determine an appropriate setting for λ, experiments were conducted under a configuration where each block contains 1000 data items, the total number of query requests is 50,000, and the sliding window size is set to 1000. Three representative access patterns were designed: (1) Stable-hot scenario, simulating a case where a small subset of high-frequency data is repeatedly accessed over a long period; (2) Uniform scenario, simulating a case where all data items have approximately equal access probabilities; and (3) Cold-hot scenario, simulating the situation where initially low-frequency data becomes frequently accessed within a short period due to sudden events.

The experimental results are presented in Fig. 11, where the Y-axis denotes the average query latency and the 99th percentile latency (P99, defined as the latency threshold not exceeded by 99% of all requests). The results indicate that: in the Stable-hot scenario, performance is optimal when $\lambda \approx 0.6$; in the Uniform scenario, the impact of λ is relatively minor, although $\lambda \approx 0.8$ yields slightly better results; and in the Cold-hot scenario, $\lambda \approx 0.4$ allows the system to adapt more rapidly to sudden changes and significantly reduces latency. Overall, when λ is chosen within the range of 0.4–0.6, the system achieves a balance between stability and adaptability, and this range is therefore adopted in the experiments.

To validate the query performance of the proposed scheme (MH_Tree), we conducted comparative analyses with schemes from the literature: Quad_Tree[17], B_Tree[18], and Merkle_Tree[24]. To simulate scenarios with varying

data scales assuming that large medical blockchain data volumes are required, the number of blocks (100, 500, 800, and 1200) in the experiment was adjusted. For each data scale, ten rounds of query experiments were performed, with 50 and 500 queries performed per round, to comprehensively evaluate the query performance. Figure 12 shows the experimental results for 50 queries per round, and Fig. 13 displays the results for 500 queries per round. The x-axis represents the round index, and the y-axis indicates the average query time per data entry.

Figure 12 shows the query rates of all the schemes under low-load query conditions. Compared with the Quad_Tree scheme, the MH_Tree scheme reduces the query time by more than 0.3 s; it shortens the query time by approximately 0.1 s relative to the B_Tree scheme and by approximately 0.15 s compared with the Merkle_Tree scheme. This trend shows that MH_Tree can effectively reduce the query cost. External factors (e.g., CPU resource occupation) may cause fluctuations in individual query times, so the analysis focuses on relative performance differences between schemes. Figure 13 shows the query rates in high-load query scenarios. The MH_Tree scheme reduces the average query time by more than 0.3 s compared to Quad_Tree, by 0.15 s compared with B_Tree, and by 0.2 s compared with Merkle_Tree. In contrast to low-load scenarios, high-load scenarios exhibit smaller query time fluctuations. This is because the increase in the number of queries smooths out the occasional fluctuations of individual queries due to factors such as uneven data distribution, reduces the interference of extreme values on the overall results, and can more accurately reflect the average performance of the solution.

From the experimental results, the MH_Tree scheme shows stable and superior query performance under different data scales and load intensities, which fully proves its practicality and efficiency in processing large-scale medical blockchain data. By introducing a weight prediction model before the data is entered into the chain, the system can effectively model and sort the medical data based on its importance, thereby realizing a data organization structure optimized by weight. This structure helps to prioritize indexing high-value data, significantly reduce the time complexity of data retrieval, and improve overall access efficiency. In addition, the aggregated Bloom filter mechanism introduced in this scheme can filter out irrelevant blocks in advance during the query process, avoiding intra-block query operations of irrelevant blocks, thereby improving query efficiency. In contrast, the Quad_Tree scheme uses a four-fork Merkle tree structure, which has the advantage of spatial division but lacks targeted data filtering; the B_Tree scheme introduces a Bloom filter, but does not achieve aggregation optimization; the Merkle_Tree scheme relies entirely on tree structure indexing, and its efficiency is limited by the level depth and path complexity.

The scheme proposed in this paper predicts data weights through a weight prediction model, stores the data in the blockchain, and accelerates data queries using the MH_Tree and aggregated Bloom filters. In comparison,

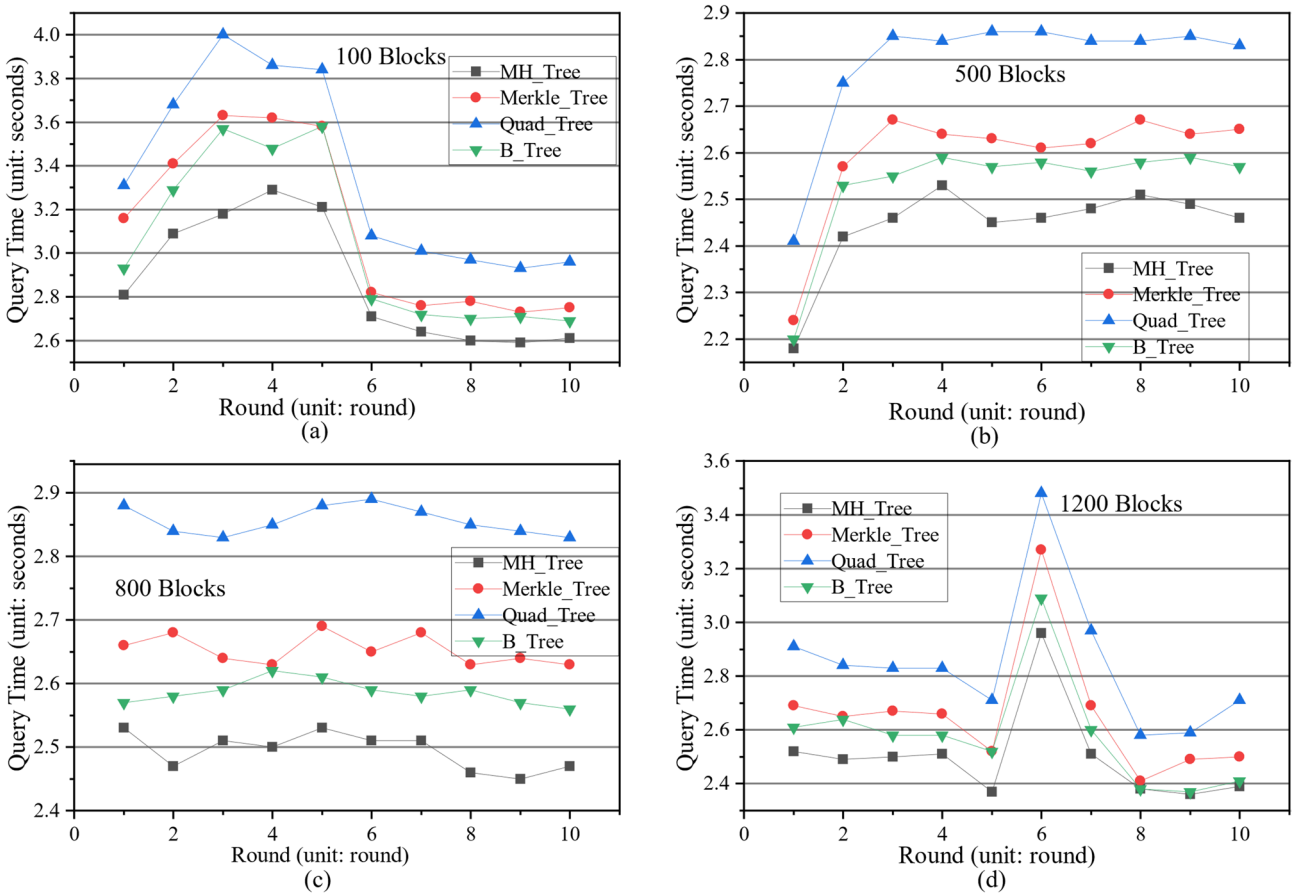


Fig. 12. 50 query results per round.

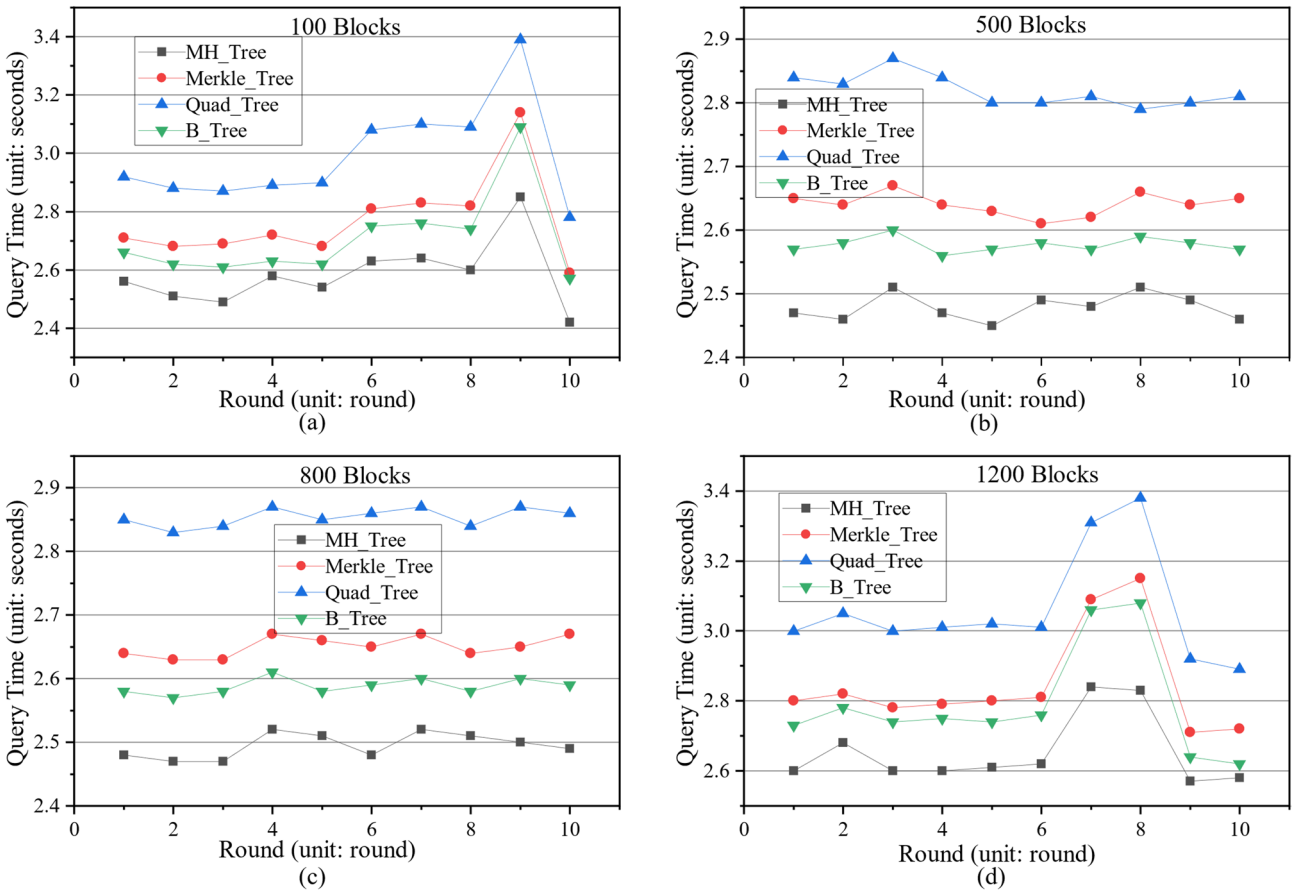


Fig. 13. 500 query results per round.

the Quad_Tree scheme increases the query speed via a quad Merkle tree structure; the B_Tree scheme combines Merkle-B trees with Bloom filters to increase the query efficiency; and the Merkle_Tree scheme accelerates queries through Merkle tree indexing. The experimental results demonstrate that our proposed query scheme, by tightly integrating the weight prediction model with aggregated Bloom filters, achieves superior query efficiency across varying query intensities. To validate the impact of the aggregated Bloom filter on the query speed, the experiment tested the query rates for identical data under two conditions: without and with the aggregated Bloom filter. In high-load query scenarios simulated with 500 queries per round across 100, 500, 800, and 1200 blocks, the comparative results of both approaches are shown in Fig. 14.

In low-data scenarios (100 and 500 blocks), query time fluctuations in some rounds were observed due to factors such as data distribution. However, across all the scenarios, the aggregated Bloom filter outperforms the non-Bloom filter approach, increasing the query speed by approximately 0.2 s. The results confirm that the aggregated Bloom filter enhances query efficiency by reducing the number of invalid queries, providing stable support for high-frequency, high-efficiency medical blockchain systems.

Additionally, the experiment compared the structural performance metrics of four schemes, including storage overhead, data insertion time, and verification path generation time, to comprehensively evaluate their applicability in medical data scenarios. The tests were conducted using 100 randomly selected data entries with 10 experimental rounds.

The average storage overhead and insertion time of the four schemes are shown in Table 1. The Merkle_Tree scheme demonstrates the most concise structure, where data is stored directly in leaf nodes without requiring additional auxiliary structures. This results in the lowest storage overhead (0.0065 MB) and shortest insertion time (0.0406 s) in the experiments. The Quad_Tree scheme adopts a quad Merkle tree structure, exhibiting comparable storage and insertion performance to the Merkle_Tree scheme at 0.0068 MB and 0.0539 s respectively. The B_Tree scheme incorporates B-tree structure and Bloom filter, where node management and bitmap updates introduce additional overhead, leading to relatively higher storage consumption (0.0089 MB) and insertion time (0.0871 s). However, it achieves superior query efficiency compared to the former two approaches.

The proposed scheme first calculates weights through the weight prediction model, then locates the insertion position in the MH-Tree, followed by updating the aggregated Bloom filter. Although exhibiting comparable insertion overhead and storage space to the B_Tree scheme (0.0085 MB and 0.0852 s), the introduction of the aggregated Bloom filter and weight optimization mechanism enables significant improvement in query efficiency in practical applications. This configuration demonstrates superior comprehensive performance, making it better suited for blockchain storage scenarios involving high-frequency access and large-scale medical data.

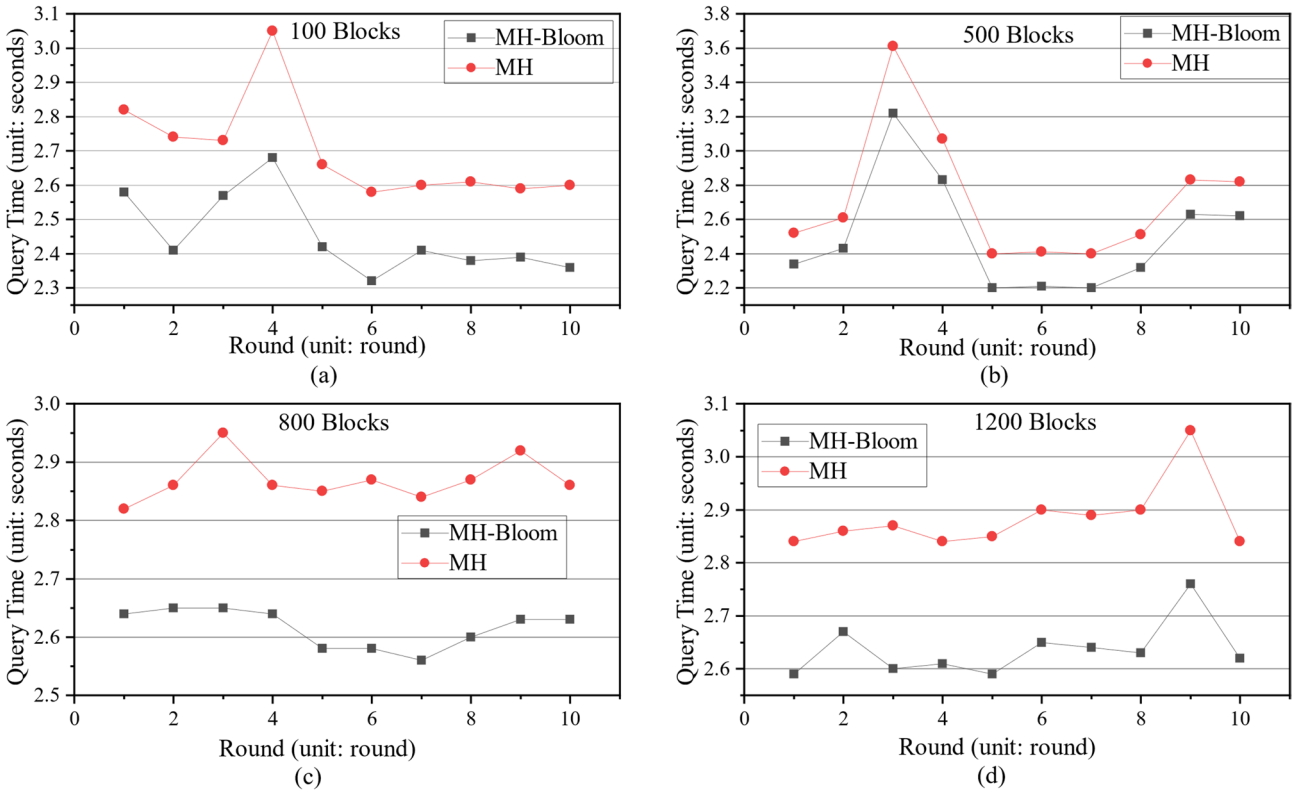


Fig. 14. Query results with and without aggregated Bloom filters.

Scheme	Average storage overhead (MB)	Average insertion time (s/record)
Proposed scheme	0.0076	0.0967
Quad_Tree	0.0068	0.0539
B_Tree	0.0089	0.0871
Merkle_Tree	0.0065	0.0406

Table 1. Comparison of average storage overhead and insertion time for different indexing Schemes.

The average verification path generation time for each scheme is shown in Fig. 15 The MH_Tree scheme achieved an average verification path generation time of approximately 0.09 s, whereas Quad_Tree required 0.04 s, B_Tree required 0.10 s, and Merkle_Tree required 0.15 s. The Quad_Tree scheme exhibited significantly faster verification path generation due to its quad-tree structure, which reduces the number of hash computations required to generate root nodes. However, quad tree structures inherently have higher search complexity than binary trees do. The MH-Tree structure results in shorter verification paths for high-frequency data, enabling faster average verification path generation than do the Merkle trees in high-volume medical data query scenarios. The B_Tree scheme, which incorporates index information in nonleaf nodes, also generates verification paths faster than Merkle trees do.

Credibility-based nonexistence proof experiment

This experiment was conducted under the following premise: light nodes receive query results indicating the absence of data and must initiate verification with other full nodes. The experiment simulated 200 verification queries using 10 full nodes with credibility values of 70. Figure 16(a) illustrates the changes in node credibility values, where the x-axis represents the query count and the y-axis indicates the credibility values of nodes used to evaluate their trustworthiness. During the experiment, the malicious behavior probability of nodes was set to 0.5%, but no malicious acts occurred. Thus, credibility fluctuations were caused primarily by the credibility decay mechanism and dynamic adjustments of the scoring system.

Figure 16(a) clearly shows that despite identical initial credibility values across nodes, their credibility values exhibited variability during the experiment. When the query count approached 150, certain nodes (e.g., Node_2 and Node_6) showed significant decreases in credibility. This occurred because these nodes received a greater proportion of query requests in these rounds, subjecting them to increased load and stricter response performance requirements, which triggered credibility adjustments. Additionally, around 170 queries, some

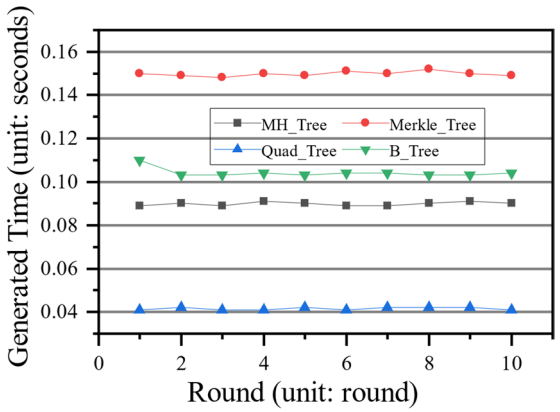


Fig. 15. Verification path generation time.

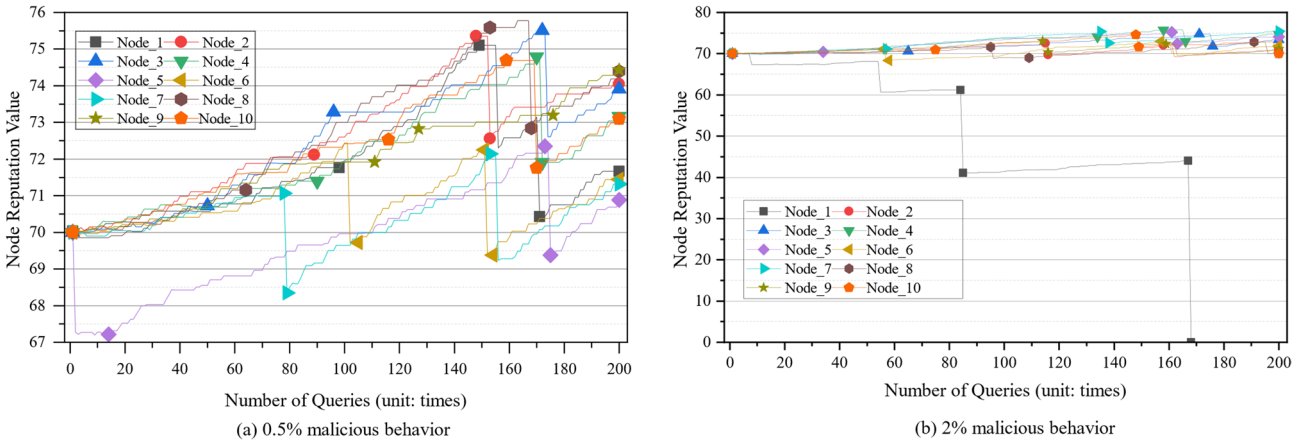


Fig. 16. Node reputation value changes at 0.5% and 2% malicious behavior probabilities.

nodes (e.g., Node_1, Node_10, and Node_5) experienced notable decreases in credibility because of the cumulative effect of credibility decay. The decline was amplified by the additional scoring adjustments in that round, building upon the accumulated credibility reductions from prior queries.

In addition to the significant changes mentioned above, the credibility values of nodes remained within small fluctuation ranges during most query rounds. These minor fluctuations arose primarily from routine adjustments by the credibility decay mechanism. Owing to slight differences in the query request distribution, task performance, and response efficiency across nodes, the scoring mechanism dynamically adjusts the credibility values on the basis of these variations, resulting in bounded fluctuations for all nodes.

The experimental results show that the node credibility values fluctuate between 67 and 76, with the overall level remaining stable at approximately 70. This finding indicates that the system effectively maintains a dynamic balance in node credibility through its scoring and decay mechanisms. Moreover, the large fluctuations observed in some nodes further validate the sensitivity and adaptive capability of the scoring system. These fluctuations demonstrate that when nodes exhibit uneven load distributions, inadequate response capacities, or cumulative decay effects, the scoring mechanism promptly adjusts their credibility values to ensure that they accurately reflect node performance.

Figure 16(b) illustrates the changes in node credibility values when the malicious behavior probability is 2%. When a node acts maliciously (e.g., refusing to provide query results or returning incorrect responses), its credibility value decreases significantly. For example, Node_1 in Fig. 16(b) committed malicious behavior during the 86th query, causing its credibility value to decrease to below 50, triggering a light penalty mechanism. In subsequent queries, even when Node_1 provided correct results, the slow recovery of its credibility value prevented rapid reputation restoration through short-term compliance, thereby ensuring system reliability. Additionally, Node_1 acts maliciously again during the 168th query. After the decrease in credibility, its value fell below 30, and the node was permanently removed from the network.

By comparing the credibility changes of nodes in two malicious behavior probability settings, the multinode collaborative verification mechanism based on credibility values can rapidly identify and address malicious actions as the probability increases. This mechanism enhances the trustworthiness of query results by integrating verification outcomes from multiple nodes while effectively mitigating the systemic impact of individual

malicious nodes. The experimental results demonstrate that collaborative verification not only strengthens error detection capabilities during data queries but also optimizes system stability and efficiency across diverse network conditions, thus providing reliable support for secure data querying and validation.

Conclusion

This paper proposes a novel efficient query scheme to address the issues of low efficiency and limitations in nonexistence proofs for lightweight medical blockchain data queries. First, by introducing the XGBoost algorithm for data weight prediction, high-weight data are prioritized for storage near root nodes in the blockchain, optimizing the storage layout and enhancing the query efficiency. Second, the aggregated Bloom filters and MH tree structure, combined with segment filtering and weight optimization, reduce query path lengths, thereby improving on-chain data query performance. Finally, to resolve the limitations of sorted Merkle trees in nonexistence proofs, a credibility-based multinode collaborative verification method is proposed using Bloom filters, ensuring query accuracy and strengthening system security.

Theoretical analysis and simulation experiments demonstrate that the proposed scheme achieves approximately 15% improvement in query efficiency compared to existing methods under medium-scale datasets (~ 55,000 records) and diverse query workloads, while maintaining enhanced system security. As the experiments primarily rely on simulated environments, certain practical factors such as data distribution and network topology fail to be comprehensively addressed, indicating that system performance in broader or more complex scenarios requires further investigation.

Given the escalating demands of medical data growth and system heterogeneity, future research will prioritize the optimization of cross-chain query mechanisms. Through the integration of lightweight consensus protocols with trusted cross-chain communication frameworks, combined with Merkle proofs and verifiable computation techniques, this approach aims to enhance both efficiency and security in cross-chain data verification. Additionally, by implementing hierarchical inter-chain relay mechanisms to optimize query pathways and reduce communication latency, this research trajectory seeks to establish a blockchain-based data query framework that demonstrates superior efficiency, security, and adaptability for resource-constrained scenarios.

Data availability

The data used to support the findings of this study are available from the corresponding author upon request.

Received: 7 March 2025; Accepted: 21 November 2025
Published online: 06 January 2026

References

1. Liu, X. et al. A systematic study on integrating blockchain in healthcare for electronic health record management and tracking medical supplies. *J. Clean. Prod.* **447**, 141371 (2024).
2. Anderson, C. et al. Blockchain innovation for consent self-management in health information exchanges. *Decis. Support Syst.* **174**, 114021 (2023).
3. Cerchione, R. et al. Blockchain's coming to hospital to digitalize healthcare services: designing a distributed electronic health record ecosystem. *Technovation* **120**, 102480 (2023).
4. Xiang, X., Cao, J. & Fan, W. Decentralized authentication and access control protocol for blockchain-based e-health systems. *J. Netw. Comput. Appl.* **207**, 103512 (2022).
5. Wang, T. et al. Health data security sharing method based on hybrid blockchain. *Future Gener. Comput. Syst.* **153**, 251–261 (2024).
6. He, C. et al. Interpretable modulated differentiable STFT and physics-informed balanced spectrum metric for freight train wheelset bearing cross-machine transfer fault diagnosis under speed fluctuations. *Adv. Eng. Inform.* **62**, 102568 (2024).
7. Samala, A. D. & Rawas, S. Transforming healthcare data management: a Blockchain-Based cloud EHR system for enhanced security and Interoperability. *Int. J. Online Biomed. Eng.* **20** (02), 46–60 (2024).
8. Kumari, D. et al. HealthRec-Chain: patient-centric blockchain enabled IPFS for privacy preserving scalable health data. *Comput. Netw.* **241**, 110223 (2024).
9. Li, J. et al. Blockchain-Based decentralized cloud storage with reliable deduplication and storage Balancing. *IEEE Trans. Netw. Sci. Eng.* **11** (4), 3289–3304 (2024).
10. Dai, X. et al. LVQ: a lightweight verifiable query approach for transaction history in Bitcoin. In *IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore* 1020–1030 (2020).
11. Yu, H. et al. EDCOMA: enabling efficient double compressed auditing for Blockchain-Based decentralized Storage. *IEEE Trans. Serv. Comput.* **17** (5), 2273–2286 (2024).
12. Yang, H. et al. CVCQ: off-chain committee-based verifiable cross-chain query scheme for large-scale BIoT. *IEEE Internet Things J.* **12** (9), 12608–12623 (2025).
13. Li, B. et al. FlexIM: efficient and verifiable index management in Blockchain. *IEEE Trans. Knowl. Data Eng.* **37** (6), 3399–3412 (2025).
14. Xiao, J. et al. Cloak: hiding retrieval information in blockchain systems via distributed query Requests. *IEEE Trans. Serv. Comput.* **17** (6), 3213–3226 (2024).
15. Zhang, H. et al. Multi-level index construction method based on master–slave blockchains. *Sci. Rep.* **14** (1), 4049 (2024).
16. Jia, D. et al. A learning-based efficient query model for blockchain in internet of medical things. *J. Supercomputing.* **80** (12), 18260–18284 (2024).
17. Liu, Z. et al. Data integrity audit scheme based on quad Merkle tree and Blockchain. *IEEE Access.* **11**, 59263–59273 (2023).
18. Sun, W., Wang, S. & Li, J. A lightweight and efficient verifiable query method for blockchain systems. *J. Chin. Comput. Syst.* **45** (08), 1944–1952 (2024).
19. Zhang, X. et al. A novel lightweight node data query method for medical blockchain. *J. Appl. Sci.* **40** (4), 600–610 (2022).
20. Cheng, J. et al. Lightweight verifiable blockchain top-k queries. *Future Gener. Comput. Syst.* **156**, 105–115 (2024).
21. Athanassoulis, M. & Ailamaki, A. BF-tree: approximate tree indexing. *Proc. VLDB Endowment* **7** (14), 1881–1892 (2014).
22. Xu, S., Guo, X. & Xu, K. Verifiable bloom filter (VBF): accelerate the query and proof of nonexistent data in a blockchain. *Sci. Sin. Inf.* **53** (12), 2386–2405 (2023).
23. Suo, J. et al. Ship fuel consumption prediction model based on XGBoost algorithm. *Navig. China*. **47** (2), 153–159 (2024).
24. Lai, C. et al. A blockchain-based traceability system with efficient search and query. *Peer-to-Peer Netw. Appl.* **16** (2), 675–689 (2023).

Acknowledgements
This work was supported by the National Natural Science Foundation of China (No. 61762046, No. 62166019) and the National Natural Science Foundation of Jiangxi Province (No. 20224BAB202019) and the Science and Technology Research Project of Education Department in Jiangxi Province (No. GJJ218505). The authors are very grateful to the reviewers for their valuable comments and to the editors for their meticulous revisions, which have significantly enhanced the quality of the paper.

Author contributions
Yunzhen Zhu and Xiaohong Deng contributed the main ideas and wrote the main manuscript. Jiayan Liu and Yijie Zou established the evaluation and provided the simulation experimental ideas. Juan Li and Yuxin Fang collected the experimental data and created the relevant charts. All the authors reviewed the manuscript.

Competing interests
The authors declare no competing interests.

Additional information
Correspondence and requests for materials should be addressed to Y.Z. or X.D.