Future Generation Computer Systems 156 (2024) 206–220

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

ANNProof: Building a verifiable and efficient outsourced approximate nearest neighbor search system on blockchain

Lingling Lu [a], Zhenyu Wen [b,c,*], Ye Yuan [d], Qinming He [a], Jianhai Chen [a], Zhenguang Liu [a]

[a] College of Computer Science and Technology, Zhejiang University, Hangzhou, China
[b] Institute of Cyberspace Security, Zhejiang University of Technology, Hangzhou, China
[c] University of Science and Technology of Chinay, Hefei, China
[d] School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China

ARTICLE INFO

Keywords:
K-ANN search
Verifiable outsourced query
Cloud computing
Blockchain
Merkle tree

ABSTRACT

Data-as-a-service is increasingly prevalent, with outsourced K-approximate nearest neighbors search (K-ANNS) gaining popularity in applications like similar image retrieval and anti-money laundering. However, malicious search service providers and dataset providers in current outsourced query systems cause incorrect user query results. To address this, we propose ANNProof, a novel framework supporting verifiable outsourced K-ANNS on the blockchain. ANNProof utilizes two innovative authenticated data structures (ADS), the Merkle HNSW node tree, and the Merkle vector identifier tree, for efficient K-ANNS query verification. Additionally, we employ the Merkle sharding tree as an ADS optimization technique, reducing the overhead of delivering verifiable queries. We implement the ADS construction protocol based on blockchain smart contracts to ensure tamper-evident datasets and enhance execution efficiency via a contract state consistency checking scheme. Extensive evaluations show that ANNProof reduces VO generation time, result verification time, and VO size by 160, 120, and 28×, respectively, compared to the state-of-the-art systems. Moreover, ADS construction using ANNProof takes at most 2% of the index construction time, resulting in a negligible overhead for implementing verifiable queries. Meanwhile, the sharding optimization accelerates ADS updates by 53×.

1. Introduction

The emergence of data-as-a-service (DaaS) technology has led to an increasing number of cloud service providers offering outsourced query services to end-users, providing access to large-scale datasets across various fields. With DaaS, users do not have to download or store the complete dataset. Instead, they can delegate queries to cloud platforms like Google Cloud [1], Amazon Web Services [2], and Microsoft Azure [3].

In this paper, we consider one of the most popular query scenarios K-Approximate Nearest Neighbors Search (K-ANNS) [4]. The outsourced K-ANNS query approach has been used in various applications, including similar images, text, music retrieval, personalized services, stock recommendations, and anti-money laundering [5–9]. K-ANNS requires a defined distance function between the vector data elements. K-ANNS aims to find the approximate K elements from the dataset that minimize the distance to a given query data element.

Example 1 (*Outsourced K-ANNS*)**.** Bob, an internet anti-money laundering commissioner, has obtained a bank account linked to a laundering ring and seeks more suspicious accounts belonging to the group [10, 11]. To achieve this, Bob may want to analyze account transaction behavior data that may be available on the data providers (DP) such as International Banking Cooperation Alliance (IBCA). Moreover, online data analytic service providers (SP) such as Demyst [12] and PeerIQ [13] can have access to these data and charge their services based on the number of queries. The IBCA maintains an account behavior dataset D, representing each element as $\langle k_i, v_i \rangle$. For instance, the exposed account ID held by Bob is k_i, while the account behavior vector is v_i. Utilizing the K-ANNS method, Bob queries the top-k elements nearest to v_i in D, discovering K accounts with behaviors most similar to the suspected account.

Challenges. Building an outsourced K-ANNS system presents the following challenges.

C1: SP delivers correct query results. SPs may provide incorrect query results for two reasons: (i) Mistakes (e.g., code errors) causing SP to produce incorrect results. (ii) Deliberate provision of incorrect

* Corresponding author.
E-mail addresses: lulingling@email.cufe.edu.cn (L. Lu), zhenyuwen@zjut.edu.cn (Z. Wen), yuan-ye@bit.edu.cn (Y. Yuan), hqm@zju.edu.cn (Q. He), chenjh919@zju.edu.cn (J. Chen), liuzhenguang2008@gmail.com (Z. Liu).
URLs: https://github.com/lulinglingcufe (L. Lu), http://www.zhenyu.info/ (Z. Wen).

https://doi.org/10.1016/j.future.2024.03.002
Received 26 October 2023; Received in revised form 26 January 2024; Accepted 1 March 2024
Available online 2 March 2024

results, either to save costs or for other obscure reasons (e.g., involvement in money laundering networks). While cross-checking results from multiple SPs could verify correctness, this approach increases query expenses and fails to address potential collusion among SPs. Consequently, a verifiable query scheme is indispensable, allowing Bob to verify the query results' correctness.

C2: DP provides tamper-evident datasets. Traditional outsourcing query models assume the DP is honest [14,15]. Nevertheless, a DP may harbour malicious intent towards SP or be compromised, delivering incorrect datasets that could lead to false accusations against SP. Therefore, enhancing DP accountability is necessary to prevent it from provisioning tampered datasets.

Our solution. This paper proposes ANNProof, a verifiable K-ANNS query framework, to tackle the above challenges. We present solutions S1 and S2, addressing challenges C1 and C2.

S1. To answer K-ANNS queries, SP constructs an index using the dataset supplied by DP. SP retrieves query results by searching a portion of the index structure and its corresponding data, denoted as I_q. If SP transmits I_q to a user, the client user can perform a local search algorithm on I_q to obtain the same K-ANNS result as SP. This study does not consider privacy leakage and follows the assumptions in [5,16], where datasets are publicly accessible. To address C1, the key is to guarantee I_q's integrity. A promising solution is to design an authenticated data structure (ADS) based on the entire index structure. Upon delivering result R to the user, SP appends I_q and the verification object VO_{sp} generated from the ADS, forming $\{R, I_q, VO_{sp}\}$. The user then verifies I_q's correctness using VO_{sp} and executes a local search algorithm to verify R.

S2. To address **C2** and prevent dataset tampering, we propose storing the hash of the dataset's associated ADS on the blockchain. If DP sends a tampered dataset to SP, the hash of the SP's constructed ADS will not match the one stored on the blockchain, allowing SP to detect DP's malicious behavior.

Ensuring the effectiveness of this solution relies on constructing unique ADS for a given dataset. We develop an *ADS construction protocol* based on blockchain to achieve this. By utilizing smart contracts, DP and SPs achieve consensus on key parameters such as indexing and ADS construction algorithms. With these parameters determined, the ADS built on a specific dataset is guaranteed to be unique. Consequently, honest SPs construct consistent ADS upon receiving the same dataset, thereby ensuring the uniqueness of the ADS hash.

Additionally, direct storage and computation of ADS on the blockchain incur significant costs, exemplified by Ethereum charging $30 for storing 1 MB of data [17]. Therefore, we propose a contract state consistency checking scheme based on an endorsement policy. This approach allows blockchain endorsing nodes to verify not the consistency of all data in the ADS but only the consistency of the ADS's Merkle root state. By doing so, the scheme reduces system load and enhances ANNProof efficiency while maintaining trustworthiness.

Empirical results show that ANNProof achieves VO generation and result verification overhead at the millisecond level, 160 and 120 × faster than the state-of-the-art systems. The sharding optimization enables a 53 × faster speed for ADS updating. ADS construction takes at most 2% of the time required for index construction, resulting in a negligible overhead for implementing the ANNProof framework.

To summarize, this paper's contributions are as follows:

- For the first time in the literature, we study the problem of outsourced approximate nearest neighbor search in the blockchain. As an initial exploration, we propose ANNProof, an efficient authentication framework for K-ANNS queries against large-scale vector datasets.
- We propose two novel ADSs, the Merkle HNSW node tree and the Merkle vector identifier tree, which support efficient K-ANNS semantic verification. Additionally, we develop the ADS optimization technique of the Merkle sharding tree, further reducing the overhead of SP delivering verifiable queries (Section 4).
- We develop the ADS construction protocol leveraging smart contracts to ensure tamper-evident dataset and enhance ANNProof efficiency using a contract state consistency checking scheme (Section 5).
- We conduct a theoretical analysis to validate the security of our proposed ADS (Section 6). Extensive experiments demonstrate the effectiveness and performance advantages of ANNProof compared with state-of-the-art systems. ANNProof is released on Github for public use [18] (Section 7).

Table 1
Notations in ANNProof.

Notation	Description
D, R	Outsourced dataset by DP, query results delivered by SP
I_q	Partial index structure searched for a specific user query
VO_{sp}	Verification objects generated from ADS to guarantee I_q's integrity
VO_{chain}	ADS's Merkle root recorded in the blockchain
Q, q	A specific user query, the queried vector
o_i, k_i, v_i	A data object in D, o_i's identifier, o_i's vector value
τ, τ_{vo}	A Merkle tree, a VO_{sp} tree
H_{index}	HNSW index
M	Number of connections between the nodes in H_{index}
$efConstruction$	Dynamic candidate list size in H_{index} construction
$efSearch$	Dynamic candidate list size in H_{index} search
sh_{size}	Maximum quantity of leaf nodes in a shard
$shid$	Sequence number of a Merkle tree shard
Id_{shid}	Visited nodes' sequence numbers in the $shid$-th Merkle tree shard
C_{ADS}	ADS construction contract
vp, μ, V_{leaf}	VP-tree's vantage point, median distance, a bucket of vectors

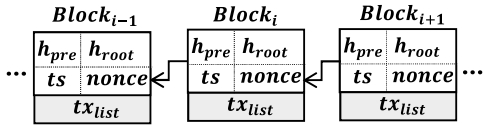


Fig. 1. Blockchain data structure.

Organization. The rest of the paper is organized as follows. Section 2 introduces some preliminaries and Section 3 describes the ANNProof design. Section 4 then presents the two proposed ADS, followed by a description of the overall ANNProof framework in Section 5. Section 6 provides a security analysis. In Section 7, we conduct extensive experiments to evaluate the performance of ANNProof. In Section 8, we reviews existing studies on outsourced query system (OQS) and compare them with ANNProof. Finally, we conclude our paper in Section 9.

2. Background and preliminaries

This section presents some background blockchain knowledge and preliminaries for verifiable queries. Table 1 summarizes the frequently utilized symbols throughout this paper.

2.1. Blockchain and smart contract

Block structure: A blockchain consists of a series of blocks sequentially linked via cryptographic hash pointers, as depicted in Fig. 1. A block comprises two components: the block body and header. The body stores a transaction list tx_{list}. The header contains a MHT root h_{root}

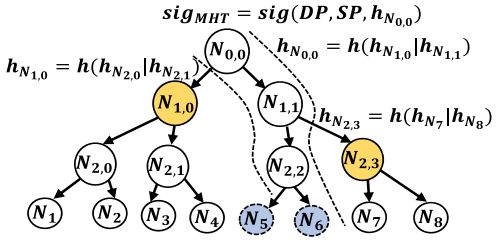


Fig. 2. An example of verifiable queries in the Merkle hash tree.

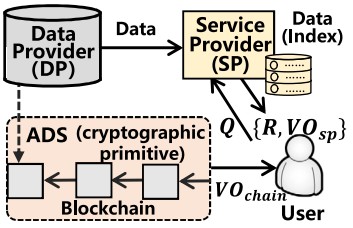


Fig. 3. Verifiable query framework in hybrid-storage blockchain.

constructed from tx_{list}, a hash of the preceding block h_{pre}, a timestamp ts, and a nonce derived from a consensus process among network peers .

Data model: The block structure is tamper-proof and serves as an immutable data source. Blockchain system adopts a key–value or account-based data model, operationalized through world state databases maintained by peers [19,20]. These databases store the key–value data, ensuring data consistency across the network.

Smart contracts: are trusted programs that can execute transactions and update the key–value data in the world state databases of peers. This functionality enhances the blockchain's ability to automate processes and enforce contractual agreements without intermediaries [21,22].

2.2. Preliminaries of verifiable outsourced queries

Outsourced queries refer to a scenario where DP outsources the data to a SP who offers query services to users [23,24]. In this context, users can utilize specific technologies for query result verification [25,26], and the Merkle Hash Tree (MHT) is a classic ADS used for this purpose [27,28].

Example 2 (*Verifiable Query using MHT*).
(1) ADS construction. SP maintains a MHT with 8 leaf nodes $\{N_1, \ldots, N_8\}$, as illustrated in Fig. 2. Each leaf node in the MHT stores a data object's digest, computed using a cryptographic hash function $h(\cdot)$. Constructed bottom-up, the MHT is a binary tree where each internal node stores a digest computed from the concatenation of its children's digests, e.g., $h_{N_{1,0}} = h(h_{N_{2,0}}|h_{N_{2,1}})$.
(2) Integrity assurance through Merkle root. MHT ensures tamper evident in two aspects. Firstly, DP and SP sign the Merkle root; thus, any alteration to $h_{N_{0,0}}$ by attackers would be detected as it would violate the DP's and SP's certificates [16]. Secondly, if attackers modify the leaf nodes in the MHT, the corresponding Merkle root, $h_{N_{0,0}}$, will adjust in response, further safeguarding the integrity of the MHT [29].
(3) VO verification. The user receives query results $R = \{N_5, N_6\}$ attached with VO_{sp} from SP. SP generates $VO_{sp} = \{h_{N_{1,0}}, h_{N_{2,3}}\}$ according to the MHT ADS. The user can then **reconstruct** a Merkle root using R and VO_{sp}, comparing it to the **publicly available** Merkle root $h_{N_{0,0}}$. If they match, the user can verify the result R is not tampered with by SP.
Verification basis. MHT-based ADS leverages the collision resistance of the hash function [30,31], ensuring that SP cannot forge data objects to make the user reconstruct a root identical to $h_{N_{0,0}}$.

2.3. Verifiable query in blockchain

Verifiable key–value queries. Blockchain technology offers publicly accessible and immutable data storage. To support verifiable key–value queries, we store data digests on-chain and maintain the complete data off-chain with SPs. In this design, the *key* refers to an identifier used to request data, and the *value* is the actual data retrieved from SPs. Users verify the integrity of query results from SPs by comparing query results' digests with the corresponding on-chain digests [23,32].

Verifiable query framework in blockchain. A hybrid-storage blockchain is proposed for supporting advanced query semantics beyond simple key–value queries, as depicted in Fig. 3. In this architecture, SP stores data off-chain, and DP uses a smart contract to maintain the on-chain ADS [16,29].

In Fig. 3, users retrieve query results $\{R, VO_{sp}\}$ from SP, and VO_{chain} from ADS in blockchain. By combining VO_{sp} and VO_{chain}, users can verify R's correctness. DP utilizes cryptographic primitives like accumulators and MHT to construct ADS [30,33,34]. When using the MHT primitive, SP stores the data locally, while DP stores the MHT root on the blockchain. Furthermore, if SP uses specific indexes to expedite query result generation, DP stores the MHT-based ADS corresponding to the index on the blockchain.

Section 3.2 will discuss the limitations of existing verifiable query frameworks in blockchain and introduce the key technologies in ANNProof that address these challenges.

3. ANNProof design and overview

This section presents the K-ANNS semantics and design ideas of ANNProof, followed by ANNProof overview and its motivation to build on blockchain.

3.1. Query semantics of ANNProof

We first formally define the K-ANNS (Top-k vector search) query approach employed in ANNProof. DP outsources dataset $D = \{o_1, o_2, \ldots, o_n\}$, consisting of n vector data objects, to SPs. Each vector data object is modeled as a tuple $o_i = \langle k_i, v_i \rangle$, where k_i serves as a unique identifier, such as the account ID in the anti-money application. v_i denotes o_i's vector value. SPs answer K-ANNS queries for users by constructing a query index based on D.

Let q be the query vector, v_n be the vector value of o_n in the dataset. We can calculate the distance between q and v_n via a distance function $d(q, v_n)$, e.g., euclidean distance, cosine distance, and dot product. To identify the k most similar vectors to q, we ensure that for $R = \{v_{top_i}\}_{i=1}^{k}$, both $\forall i \in [1, k]$ and $\forall v_n \notin R$ satisfy Eq. (1).

$$d(q, v_n) \geq d(q, v_{top_i}) \tag{1}$$

3.2. Key technologies of ANNProof

The existing blockchain-based verifiable query framework, discussed in Section 2.3, does not satisfy verifiable K-ANNS queries for two primary limitations:

1. **K-ANNS verification issue**: Conventional studies use a tree structure for the off-chain data index [5,6,31]. However, employing tree-based approaches like VP-tree or k-d tree for K-ANNS semantics leads to significant user verification overhead.

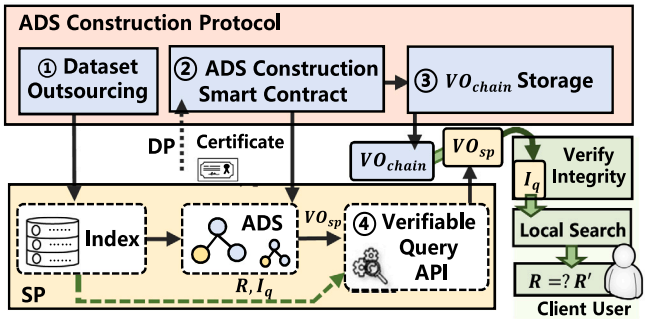


Fig. 4. ANNProof overview based on the ADS construction protocol.

2. **ADS construction cost issue**: The economic cost of computing and storing complex ADS on public blockchains is substantial [17].

To address the verification of query semantics, we first design a verification workflow and then propose two ADSs.

- Verification workflow of K-ANNS: Users obtain partial index I_q from SP and perform a local search process on I_q to obtain the K-ANNS result. User verification cost aligns with the computational complexity of SP searching the complete index to generate the K-ANNS result. In this workflow, the hierarchical navigable small world graphs (HNSW) index outperforms other K-ANNS indexes due to the efficiency of its search algorithm, providing users with a **smaller size** I_q, shorter search time, and **determined search paths**.
- ADSs: We propose the Merkle HNSW node tree as ADS to ensure the integrity of each node in I_q, thereby guaranteeing the overall integrity of I_q. Besides, users usually query SP using vector identifiers, as illustrated in Example 1 of Section 1, where Bob uses the account ID as the query attribute. To prevent potential dishonesty from SP via incorrect account vectors, we propose the second ADS, the Merkle vector identifier tree, which ensures the integrity of identifier-vector relationships.

To mitigate the construction costs of ADS, we store the ADS root on the blockchain instead of the complete ADS. Besides, EVM (ethereum virtual machine) in public blockchain lacks support for executing complex ADS construction [35], leading us to opt for permissioned blockchain and endorsement policy to ensure the trustworthiness of the ADS construction process. Additionally, we employ sharding techniques to transform the ADS into a smaller-sized MHT, reducing the computation time for SPs to generate VO. The tradeoff here lies in increasing VO_{chain} numbers stored on the blockchain to decrease the computation overhead on SPs.

3.3. ANNProof overview

Fig. 4 illustrating the overall ANNProof workflow based on the ADS construction protocol, comprising four steps.

Step ①. DP outsources datasets to SPs. Each SP constructs an index using the dataset to answer K-ANNS queries efficiently.

Step ②. To address C2, DP uses the identity certificate to invoke the ADS construction smart contract, pre-deployed on the SP, to prevent dataset tampering. Given pre-defined construction algorithms and parameters, SPs can locally build consistent ADS based on the same dataset.

Step ③. We store VO_{chain} on the blockchain as the execution result of the ADS construction smart contract. This allows users to verify outsourced queries using the publicly available VO_{chain}, as demonstrated in Example 2 and Fig. 3.

Step ④. SP offers a verifiable query API to users. Given a user query Q, SP searches the index to obtain I_q and R, then generates VO_{sp} for I_q based on the ADS. In Fig. 4, upon receiving $\{R, I_q, VO_{sp}\}$ from SP, the client user retrieves VO_{chain}. By combining VO_{chain} and VO_{sp}, users can verify I_q's integrity and subsequently execute a local search algorithm to obtain R', thus verifying the consistency between R' and R.

ADS update protocol. When DP inserts new vectors into the dataset, SP must update the ADS correspondingly. We design an efficient *ADS update protocol* that enables SP to enhance efficiency by incrementally constructing the ADS rather than fully reconstructing it (more details in Section 5.3).

Security requirements of verifiable queries. Using VO, users can verify the query results' soundness, completeness, and freshness, meeting the following security requirements.

- **Freshness**: All answers are derived from the most up-to-date version of the dataset.
- **Completeness**: The query results are guaranteed to include all valid answers without omissions.
- **Soundness**: All answers satisfy the query criteria and are exclusively sourced from the designated DP.

Section 6 provides the security analysis that formally proves ANNProof can meet the above-mentioned requirements.

3.4. Motivation for implementing ANNProof on blockchain

The adoption of blockchain technology in ANNProof is driven by the need to detect malicious activities potentially conducted by DP and SPs effectively. When outsourcing datasets to SPs, DP stores the hash of the dataset's associated ADS on the block-chain. Once DP commits the hash value to the blockchain, it is public and immutable. This mechanism, illustrated in Section 3.3, Fig. 4, operates via an smart contract-based ADS construction protocol.

Detection of malicious activities by DP. If a DP sends a *tampered* dataset to SPs, a discrepancy between the ADS hash constructed by the SP and the hash recorded on the blockchain will be evident, signaling potential DP misbehavior.

Detection of malicious activities by SP. When a SP provides *erroneous* $\{R, I_q, VO_{sp}\}$ to users, such incorrectness can be identified by users through the combined use of VO_{sp} and VO_{chain}, as described in Section 3.3, step ④.

4. Verifying K-ANNS query via Merkle HNSW node tree

This section presents the Merkle HNSW node tree that implements the ANNProof workflow ① ∼ ④ delineated in Section 3.3, demonstrating how SP constructs query index, builds ADS and delivers verifiable queries within the corresponding steps. Subsequently, the Merkle vector identifier tree is introduced, followed by a formal complexity analysis of the Merkle sharding tree that demonstrates the efficiency of this ADS optimization technique.

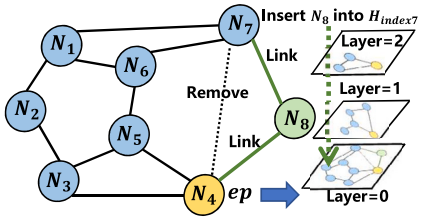


Fig. 5. Insert the 8-th vector, denoted as N_8 into the HNSW index.

4.1. Query index construction

As described in Section 3.3 step ①, SP first needs to build an index for the upon receiving dataset D from DP. We employ Algorithm 1 to construct a hierarchical navigable small world graphs (HNSW) index [36].

Index construction overview. With each vector insertion, a node, defined in Eq. (2), is added to H_{index}. The identifier of the next inserted node is $id + 1$. Algorithm 1 updates N_{id}'s neighborhood $eConn$. Upon inserting all vectors into the HNSW index, the construction of H_{index} is finalized, comprising n nodes.

$$N_{id} = \{id, v_{id}, eConn\} \tag{2}$$

Algorithm 1: INSERT(v_{id}, $H_{index_{id-1}}$)

Input: v_{id}, $H_{index_{id-1}}$, N_{id}, M, $efConstruction$, seed
Output: $H_{index_{id}}$

1 $l_c \leftarrow$ GetLevel(seed, v_{id})
2 $ep_{l_c} \leftarrow$ GetEp($H_{index_{id-1}}.ep$, l_c)
3 $neighbors \leftarrow$ Search-Layer($v_{id}, ep_{l_c}, efConstruction, l_c$)
4 $N_{id}.eConn[l_c] \leftarrow neighbors$
5 Update-Neighbors($neighbors, v_{id}, M, l_c$)
6 $H_{index_{id}} \leftarrow$ Append($H_{index_{id-1}}, N_{id}$)

Vector insertion algorithm. Algorithm 1 takes the following inputs: the id-th inserted vector v_{id}, the pre-insertion index $H_{index_{id-1}}$, the inserted node N_{id}, the number of connections M between the node and other nodes, the dynamic candidate list size $efConstruction$, and the seed for generating level of v_{id}. The output is the updated index $H_{index_{id}}$ [37]. N_{id}'s level l_c is generated from the seed (line 1). By using the enter-point in l_c (line 2), the $Search-Layer$ function locates v_{id}'s neighbors (lines 3). These neighbors are placed in N_{id}'s $eConn$ (line 4). Afterward, $Update-Neighbors$ function updates $neighbors$ by adding id to their $eConn$ (line 5). Lastly, N_{id} is appended to $H_{index_{id-1}}$, yielding the updated index $H_{index_{id}}$ (line 6).

Example 3 demonstrates the HNSW index construction, highlighting vectors' individual insertion and multiple neighbor relationships in index nodes.

Example 3 (H_{index_8} *Construction*)**.** SP constructs the HNSW index by sequentially inserting each vector into the index. SP employs Algorithm 1 to insert the 8-th vector into H_{index_7}. Fig. 5 illustrates a three-layer graph index, denoted as H_{index_7}. In each layer, nodes connect to their neighboring nodes. With $M = 3$, a node can have a maximum of 3 neighbors with minimized vector distance.

First, N_8's layer level, denoted as $l_c = 0$, is generated from the seed (line 1). The enter-point in layer 0 is $ep_0 = N_4$ (line 2). The $Search-Layer$ function starts from N_4, which connects to N_3, N_5, and N_7. So we calculate the distance between these neighbors and v_8. With $efConstruction = 6$, we continue to calculate the distance between N_3's neighbor (N_2), N_6's neighbor (N_1, N_7) and v_8, choosing 3 nearest neighbors of v_8 (line 3). Then we link N_8 with N_4 and N_7 (line 4), updating N_4 and N_7 by removing the edge between them (line 5). Upon inserting N_8 into the graph, SP finishes H_{index_8} construction (line 6).

Advantages of HNSW. We explain why HNSW is a promising solution for outsourced query scenarios from two aspects.
(1) General semantic. ANNProof employs K-ANNS for query semantics, a versatile method for vector similarity retrieval in domains such as image, text, and music [7,8]. For example, by representing songs as multi-dimensional vectors, K-ANNS efficiently retrieves similar music using HNSW. In contrast, prior work like ImageProof is limited to image retrievals using k-d tree and inverted index [5].
(2) Performance superiority. Two categories of indices can support K-ANNS: tree indices and graph indices, with VP-tree [38] and HNSW as respective examples. Addressing C1 requires maintaining the integrity of I_q, with smaller sizes yielding better verifiable query performance. HNSW outperforms VP-tree in search efficiency by requiring fewer nodes for a given query q, thus reducing I_q size.

- VP-tree uses a vantage point to partition the search space based on other nodes' distances to this vantage point [39]. In high-dimensional spaces, the sparsity of data node distribution may increase the number of nodes to be searched.
- HNSW employs a **multi-layered graph** structure, enabling quicker proximity to the target node during the search. The **layered approach** assigns data nodes to different graph layers, each with unique neighbors at different layers. Upper layers contain fewer nodes than lower layers, so a search initiating from the top can swiftly locate an approximate nearest neighbor, followed by more detailed searches in lower layers, substantially amplifying search efficiency.

4.2. ADS construction: Merkle HNSW node tree

To ensure the integrity of I_q, a subset of H_{index} nodes, we use MHT-based ADS, with index nodes as MHT leaf nodes (Section 3.3 step ②). However, as the number of index nodes increases, VO generation complexity exponentially increases. This section introduces a novel ADS called the sharded Merkle HNSW node tree, which partitions ADS into sub-trees, limiting the complexity of ADS to a sharding threshold and reducing the VO generation complexity.

ADS data structure. The leaf node digest is obtained using Definition 1, followed by constructing the internal node using Definition 2. As described in Section 3.3 step ③, the ADS's Merkle root will be stored in the blockchain.

Definition 1 (*Digest of Leaf Node*)**.** N_{id}, denoted by sequence number id, is a HNSW index node (also a node in I_q). Eq. (3) denotes $h_{N_{id}}$ as node information's hash comprising N_{id}'s three components, delineated in Eq. (2). "|" is the string concatenation operator.

$$h_{N_{leaf}} = h_{N_{id}} = h(h_{id}|h_{v_{id}}|h_{eConn}) \tag{3}$$

Definition 2 (*Digest of Internal Node*)**.** The internal node digest is defined in Eq. (4). $h_{N_i^l}$ ($h_{N_i^r}$) is the digest of N_i's left (right) child.

$$h_{N_i} = h(h_{N_i^l}|h_{N_i^r}) \tag{4}$$

Merkle sharding tree optimization. By partitioning the ADS into multiple shards, SP reduces the size of each sub-Merkle tree and minimize verification costs for outsourced queries. With n index nodes in H_{index} and a maximum leaf node allowance per shard denoted as sh_{size}, Eq. (5) defines the shard quantity S_n. Besides, τ_{shid} denotes the $shid$-th Merkle tree shard as shown in Fig. 6.

$$S_n = \lceil \frac{n}{sh_{size}} \rceil \tag{5}$$

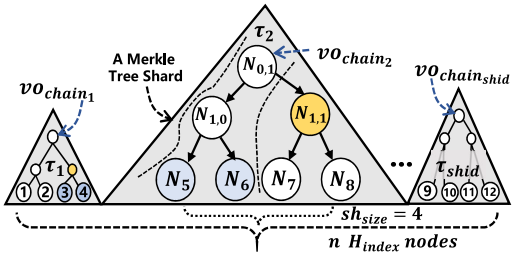


Fig. 6. Sharded Merkle HNSW node tree.

Example 4 (*Conversion of Unsharded Tree to Sharded Tree with Optimization Techniques*)**.** Fig. 2 depicts an unsharded tree with 8 leaf nodes. N_5's information includes id 5, its vector, and neighbor relations (N_3, N_4, and N_6 links in Fig. 5). The leaf digest h_{N_5} is the hash of N_5's information. Fig. 6 illustrates a sharded tree with $sh_{size} = 4$, partitioning 8 leaf nodes into 2 subtrees, each containing 4 nodes. τ_2 is the second Merkle tree shard with root VO_{chain_2}. As more nodes are added to the ADS, new shards, like τ_{shid} (comprising 4 nodes with root $VO_{chain_{shid}}$), must be constructed.

4.3. VO Generation and Results Verification

SP generates VO, and users verify VO in the outsourced query system. We detail each algorithm in this section.

Algorithm 2: VO Generation (by SP).

Input: q, H_{index}, K, $efSearch$, Merkle HNSW node tree shard τ_{shid}
Output: VO_{sp}

1 $W \leftarrow \emptyset$ // currently found nearest elements to q
2 $L \leftarrow$ get level of $H_{index}.ep$
3 **for** $l_c \leftarrow L...0$ **do**
4 $\quad W, I_q \leftarrow$ Search-Layer$(q, H_{index}.ep, efSearch, l_c)$
5 $R \leftarrow K$ nearest elements from W to q
6 Id_{shid}= GetNodeID$(shid, I_q)$ ▷ Step 1
7 VO_{sp} = MHT-VO-Generate(τ_{shid}, Id_{shid}) ▷ Step 2

VO generation of SP. Algorithm 2 summarizes the process of VO generation. Firstly, upon receiving user query q, SP follows the search algorithm to traverse H_{index}, obtaining $\{R, I_q\}$ as query results. Secondly, SP generates VO_{sp} for I_q based on the Merkle HNSW node tree ADS.

(1) K-ANNS stage (lines 1–5). Algorithm 2 requires the following inputs: query element q, H_{index}, number of nearest neighbors to return K, and the dynamic candidate list size $efSearch$. The search procedure initiates at the top layer L and proceeds to the bottom layer (line 3). The $Search-Layer$ function incorporates the element nearest to q at the current level l_c into W and visited index nodes into I_q (line 4). Finally, we add the top-k neighbors into R (line 5).

(2) VO generation stage (lines 6–7). The nodes within I_q are distributed across multiple Merkle tree shards ($\tau_1, \tau_2, \ldots, \tau_{shid}$) in Fig. 6. Therefore, to guarantee the integrity of I_q, SP must generate a VO_{sp} for each shard. In Algorithm 2, we demonstrate the generation of VO_{sp} within a shard by using the $shid$-th tree shard, τ_{shid}, as an example. First, we must obtains Id_{shid} from I_q via the $GetNodeID$ function (line 6). Id_{shid} denotes the visited nodes' sequence number in the $shid$-th shard, we can calculate it via Eq. (6).

$$I_q = \bigcup_{shid=1}^{shid=S_n} \{N_{id} | id \in Id_{shid} \wedge (shid-1) \times sh_{size} < id \leq shid \times sh_{size}\} \quad (6)$$

In step 2, SP generates VO_{sp} via the $MHT-VO-Generate$ function (line 7). Given that the Merkle HNSW node is also an instance of the MHT, SP can employ the VO generate function analogous to the one described for MHT in Section 2.2 and [27].

Example 5 (*Generating VO Using Algorithm 2 Based on ADS in Fig. 6*)**.** Upon receiving user query q, SP applies the K-ANNS algorithm to the HNSW index shown in Fig. 5. The $Search-Layer$ function adds visited nodes to I_q, with $I_q = \{N_3, N_4, N_5, N_6\}$ (line 4). SP then retrieves $R = \{N_5\}$ (line 5). In the next stage, SP generates VO_{sp} for I_q based on the sharded Merkle HNSW node tree as shown in Fig. 6. After executing $Id_2 = GetNodeID$ (2, I_q), we get $Id_2 = \{5, 6\}$, meaning the visited nodes' sequence numbers in the second shard are $\{5, 6\}$ (line 6). Hence, we generate $VO_{sp} = \{h_{N_{1,1}}, h_{N_5}, h_{N_6}\}$ based on the second MHT shard (τ_2) (line 7).

Algorithm 3: VO Verification (by client user).

Input: VO_{sp} , VO_{chain}, R, I_q, q
Output: *True* if result R is valid, *False* otherwise.

1 Check(VO_{chain} == ConstructRoot(VO_{sp}, I_q)); ▷ Step 1
2 Check(R == SearchGraph(I_q, q)); ▷ Step 2

VO verification of users. The following illustrate how users verifies query result R using VO_{sp} and VO_{chain} via Algorithm 3 (Fig. 4 step ④). If R is correct the algorithm will output true.

Step 1: the user reconstructs a Merkle root with VO_{sp} and I_q, checking its consistency with VO_{chain}.

Step 2: the client user locally executes the search algorithm described in the K-ANNS stage to retrieve query results R' based on I_q. Instead of obtaining the entire H_{index} from SP, the user can retrieve only the portion of the index structure required for user query's search process. As step 1 ensures the integrity of I_q, R' obtained via the *SearchGraph* function can verify that SP has delivered the correct query result if $R = R'$.

4.4. Merkle vector identifier tree

Problem. In Example 1 of Section 1, Bob seeks accounts exhibiting behavior similar to the one with account ID k_i. Elements in the dataset are denoted as $\langle k_i, v_i \rangle$, with k_i representing the account ID and v_i the behavior vector. Bob uses k_i, not the vector itself, to request an outsourced query. *If SP deceives Bob by providing a v_i that does not match k_i, Bob obtains incorrect query results.* Consequently, we present the second ADS, the Merkle vector identifier tree, designed to safeguard the integrity of identifier-vector (k_i, v_i) relationships.

(1) ADS data structure. SP builds the ADS based on dataset $D = \{o_1, o_2, \ldots, o_n\}$, where $o_i = \langle k_i, v_i \rangle$. In Eq. (7), a leaf node digest of ADS is a o_i's hash, consisting of the identifier's hash h_{k_i} and vector value's hash h_{v_i}. The internal node is stated in Definition 2.

$$h_{N_{leaf}} = h_{o_i} = h(h_{k_i} | h_{v_i}) \quad (7)$$

(2) VO generation and verification. Given that the Merkle vector identifier tree is also an instance of the MHT, SP can employ the VO generate function analogous to the one described for MHT in Section 2.2 and [27]. Similarly, the client user's query verification technique corresponds to the approach described in Example 2.

4.5. Complexity analysis of the sharding optimization

The Merkle sharding tree technique enhances the efficiency of both ADSs defined in Sections 4.2 and 4.4. It partitions the MHT-based ADS into multiple *shards*, reducing tree size and VO construction cost. Subsequently, we formally analyze the complexity optimization derived from the sharding technique.

(1) Merkle sharding tree construction cost. Given the leaf node size n, a fully-structured Merkle sharding tree consists of $\lceil \frac{n}{sh_{size}} \rceil$ shards, each containing sh_{size} leaf nodes. Let F represent the Merkle sharding tree's

fanout, r denotes the leaf node input, d_{merkle} indicates the tree's height, and h symbolizes a hash value. Merkle sharding tree's construction cost is contingent upon the hash function computations and the total I/O operations. First, constructing the leaf level necessitates sh_{size} hash computations with an input length of $|r|$. $C_{H_{|r|}}$ represents the cost of hashing a length input $|r|$ and $C_{IO_{|h|}}$ denotes the cost of storing h. Furthermore, a hash with an input length of $F \cdot |h|$ is calculated for each internal node. $C_{H_{F\cdot|h|}}$ represents the cost of hashing the length input $F \cdot |h|$. Considering the tree height $d_{merkle} = \log_F sh_{size}$, we can compute the average number of internal nodes via $N_I = \frac{F^{d_{merkle}}-1}{F-1}$. Therefore, Eq. (8) defines the cost of constructing the Merkle sharding tree.

$$C_{merkle}^{construct} = \lceil \frac{n}{sh_{size}} \rceil (sh_{size} \cdot (C_{H_{|r|}} + C_{IO_{|h|}}) + N_I \cdot (C_{H_{F\cdot|h|}} + C_{IO_{|h|}})) \tag{8}$$

(2) VO construction cost. Eq. (9) denotes the VO construction cost of processing a query that contains a single leaf node. In Eq. (9), d_{merkle} equals the number of boundary nodes contained in VO_{sp}.

$$C_{merkle}^{vo} = C_{IO_{|r|}} + d_{merkle} \cdot C_{IO_{|h|}} \tag{9}$$

(3) VO verification cost. Eq. (10) denotes a client user's VO_{sp} verification cost of a query that contains a single leaf node.

$$C_{verify}^{vo} = d_{merkle} \cdot C_{H_{F\cdot|h|}} + C_{H_{|r|}} + C_{merkle}^{vo} \tag{10}$$

Analyses. Upon observation of Eqs. (9) and (10), it becomes evident that d_{merkle} is a critical parameter impacting both the construction and verification costs of VO_{sp}. For a sharded Merkle tree, $d_{merkle} = \log_F sh_{size}$, whereas for a non-sharded Merkle tree, $d'_{merkle} = \log_F n$. $d'_{merkle} > d_{merkle}$, thus leading to the conclusion that the sharding technique enhances the efficiency of ADS.

5. Implementation of ANNProof on blockchain

This section initially presents an efficient contract execution scheme, followed by a detailed exposition of the steps involved in the ADS construction protocol. Subsequently, we extend this protocol to the ADS update protocol, which substantially enhances ANNProof efficiency.

5.1. The contract execution scheme

We develop ANNProof on the blockchain, primarily by recording the ADS hash on the blockchain via smart contracts, thereby addressing C2. Concurrently, two efficiency-related design challenges arise when implementing a trustworthy execution scheme for smart contracts.

- Efficiency of Execution (**Q1**): Does it necessitate the execution of the ADS construction smart contract by all nodes to ensure ADS uniqueness? In public blockchains, all nodes execute the same ADS construction contract, with the majority of nodes validating the correct contract execution outcome to guarantee ADS uniqueness. However, this approach leads to redundant computations and wastage of node resources.
- Trustworthiness of Execution (**Q2**): Should the complete ADS be stored on the blockchain to prevent tampering? Unfortunately, storing and processing vast amounts of data on the blockchain incurs high costs and impairs the blockchain performance [29].

To address these challenges, we propose a contract state consistency checking scheme based on an endorsement policy. We then describe the scheme in detail, illustrating its capacity to enhance efficiency while ensuring trustworthy execution.

Endorsement policy. The endorsement policy delineates the requisite nodes or their quantity to vouch for a smart contract's correct execution [40]. By defining which nodes should execute the ADS construction contract to ensure ADS uniqueness, the endorsement policy prevents unnecessary duplication of contract execution across all blockchain nodes, thereby minimizing redundant computations among nodes and **solving Q1**. A custom endorsement policy permits the smart contract to designate endorsers as a subset of necessary nodes, employing monotone logical expressions on sets, such as "two out of three" or $(A \wedge B) \vee C$.

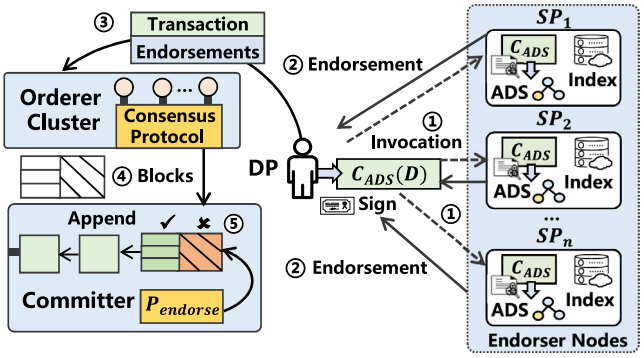


Fig. 7. Workflow of invoking and executing the ADS construction contract.

Contract state consistency checking. In ANNProof, storing the complete ADS on the blockchain is unnecessary. Instead, storing only ADS's Merkle root (VO_{chain}) on the blockchain ensures tamper evidence and guarantees the integrity of the complete ADS. This approach eliminates the wastage of blockchain storage and **answers Q2**. Regarding contract state consistency checks, we solely checks the consistency of the contract execution result, specifically the VO_{chain} state, after executing the ADS construction contract (C_{ADS}). As the ADS is an instance of the MHT, ensuring the uniqueness of the ADS's Merkle root guarantees that different SPs maintain identical ADS locally, resulting in the same ADS roots.

5.2. ADS construction protocol

The ADS construction protocol serves as the foundation for implementing the ANNProof framework. We illustrate the contract execution workflow in Fig. 7.

Step 1: Dataset Outsourcing. First, DP and multiple SPs negotiate the query index's construction algorithm and parameters. Given predefined index construction algorithms and parameters, SPs can build a determined index on the same dataset. Then, DP outsources D to multiple SPs. SPs build local query indexes using D to answer user K-ANNS queries efficiently.

Step 2: ADS Construction Smart Contract.
Preparations. Prior to contract execution, we need to make some blockchain network preparations [40]. DP and multiple SPs negotiate, employing certificates issued by the blockchain certificate authority (CA) [41,42] to sign a smart contract endorsement policy, denoted as $P_{endorse}$. Both parties must confirm the ADS construction method and parameters in C_{ADS}. C_{ADS} is then deployed on each SP, with $P_{endorse}$ = "two out of three".
① *Contract invocation*. First, DP invokes C_{ADS} as shown in Fig. 7. As stated by $P_{endorse}$, three SPs act as endorsers nodes, and DP signs and sends multiple ADS construction transactions to these SPs.
② *Contract execution endorsement*. Upon receiving the transaction request, SPs (endorsers) construct ADS from the local query index via C_{ADS} and subsequently return ADS's Merkle root as the endorsement to DP.
③ *State consistency checking*. Employing the contract state consistency checking scheme, DP must gather the VO_{chain} endorsement from two of the three SPs and verify the consistency of their values to ensure the

validity of the transaction execution result. During this process, VO_{chain} will contain the signatures of DP and SPs. Finally, DP transmits the transaction and SP endorsements to the Orderer cluster.

Step 3: VO_{chain} Storage. Next, we describe the workflow of committing transactions to the blockchain.

④ *Transactions ordering*. Orderer clusters in Fig. 7 utilize consensus protocols, such as PBFT [43,44], BFT [45,46], and XFT [47], to batch transactions into blocks and distribute them to committer nodes.

⑤ *Transactions committing*. Committer nodes validates transaction legitimacy, including transaction signature integrity, endorsement policy adherence, and read-write set version consistency with the blockchain [19]. Subsequently, valid transactions within the block are appended to the blockchain, and VO_{chain} states are updated accordingly.

5.3. ADS update protocol

In scenarios where DP adds new vectors to the dataset, the ADS construction protocol's efficiency can be further enhanced. Therefore, we extend the ADS construction protocol to the ADS update protocol comprising three similar steps.

Motivation. Take the account transaction behavior dataset as an example; DP needs to insert new account behavior vectors as the number of accounts grows, necessitating SP to update the ADS. To enhance the efficiency of updating ADS, SP is not obligated to reconstruct the entire ADS when nodes are inserted to H_{index}. Instead, SP only needs to reconstruct the corresponding ADS incorporating the updated nodes.

Step 1: Dataset Outsourcing. Assuming the original dataset contains n_1 account vectors and the account number post-update is n_2, DP needs to transmit n_2 vectors to SPs. As HNSW is an incremental index data structure defined in Section 4.1, reconstructing the entire index is unnecessary. Instead, the original index is updated with new index nodes.

Step 2: ADS Construction Smart Contract. SP constructs the updated ADS using the updated H_{index} via C_{ADS}. Taking the Merkle HNSW node tree as an illustration, this ADS incorporates the sharding optimization, distributing index nodes in different Merkle shards. Thus, there are **two cases**: (i) SP updates the original ADS, and (ii) SP creates new ADS.

Case 1: Some pre-existing index nodes' neighborhoods may change when incorporating new nodes into H_{index}. Therefore, SP only reconstructs the ADS for the corresponding Merkle tree shards housing these nodes.

Case 2: Since new nodes have been added to H_{index}, SP solely constructs new Merkle shards to accommodate these nodes. Regarding the Merkle vector identifier tree, SP only constructs new Merkle shards based on these newly added vector nodes.

Step 3: VO_{chain} Storage. If SP updates the original ADS, the corresponding VO_{chain} field (ADS's Merkle root) recorded on the blockchain will be updated. If SP creates new ADS, the corresponding new $VO_{chain_2}, VO_{chain_3}, \ldots VO_{chain_n}$ will be recorded on the blockchain.

6. Security analysis

Definition 3. We define the query results of ANNProof as **fresh**, **complete**, and **sound** when an adversary SP, limited to polynomial-time computational power, delivers results that users have a negligible probability of successfully validating.

An adversary SP, denoted as SP_{adv}, aims to forge VO_{sp} to pass the user's validation of the query result. First, SP_{adv} constructs ADS and the corresponding VO_{chain} via C_{ADS}. Second, SP_{adv} receives a query Q, generating the result $R = \{o_1, o_2, \ldots, o_K\}$, and VO_{sp}. The attack is considered successful if VO_{sp} passes result verification and any of the following conditions is true.

Condition 1 **Stale**: $(o_* \in R) \wedge (o_* \in D_0) \wedge (o_* \notin D_1)$.

Condition 2 **Incomplete**: $(o_* \notin R) \wedge (o_* \in R' \wedge R' = Q(D_1))$.

Condition 3 **Incorrect**: $(o_* \in R) \wedge (o_* \notin R' \wedge R' = Q(D_1))$.

Theorem 1. *The verifiable query algorithms of ANNProof satisfy the security property of Definition 3, contingent upon the underlying hash function demonstrating collision resistance.*

Proof. We prove this theorem by contradiction.

Case 1: Stale or incorrect results assume o_* exists within R. Stale results imply that o_* derives from the previous dataset version D_0, while incorrect results indicate its absence from the latest dataset version D_1. The client user reconstructs the Merkle root of MHT-based ADS, containing o_*, and checks consistency with VO_{chain}. A tampered result implies two different Merkle trees with the same hash root, indicating a successful collision of the underlying hash function, which contradicts our assumption.

Case 2: Incomplete results assume a valid answer is missing from R. The client obtains R' by executing the search algorithm using I_q for Q's search process. Completeness is confirmed if $R = R'$. Integrity corruption in the partial index I_q leads to hash collisions in some sharding trees of the MHT-based ADS, contradicting the assumption.

Case 3: Incorrect results assume R contains an object o_* not satisfying query Q. Completeness proof demonstrates the impossibility of such a case, as the user verifies Q's valid answer R'. R is considered complete and sound only if $R' = R$, i.e., o_* satisfying Q.

7. Evaluation

In this section, we provide a set of comprehensive experiments. The experimental results show that ANNProof reduces VO generation time, VO verification time, and VO size by 160, 120, and 28 ×, respectively, compared to the state-of-the-art methods. Although ANNProof incurs increased building overhead in the ADS construction protocol compared to the Baseline, it achieves higher recall, thereby enabling more precise user queries. By using the Merkle HNSW node tree in the ADS update protocol, ANNProof achieves two times faster than Baseline in ADS updating.

7.1. Experimental setup

Cluster setup. We build a permissioned blockchain network utilizing Hyperledger Fabric v2.2 [48] as the foundational block-chain technology deployed on a cluster with 7 nodes. The cluster is interconnected through a 1 Gbps network. Each node has identical specifications, including a 3.00 GHz Intel Xeon E3 v6 CPU with 32 cores, 32 GB RAM, a 2 TB hard drive, and the Ubuntu 20.04.4 Trusty operating system.

Implementation. We write the smart contract in the Go language. We employ an endorsement policy adhering to a "three out of four" requirement, signifying that consistent endorsement must be provided by at least three out of the four SPs. We developed a variant of the nmslib library [49], referred to as v-nmslib [18], which SP employs to construct indexes and ADS on individual nodes. The query user operates on a singular node. We implement the query processing and the result verification programs in C++. We also use the Keccak-256 [50] cryptographic hash function for VO generation. We conduct verifiable top-10 ANNS queries on 1000 randomly selected vectors and calculate the average results to obtain the experimental data. In summary, ANNProof consists of 1200 lines Go code and 3000 lines C++ code, which is publicly accessible [18].

Datasets. Our experiments were conducted on 3 widely used datasets. The scale-invariant feature transform (SIFT) is a local feature descriptor utilized in computer vision for numerous tasks, such as image retrieval [51], comprising one million 128-dimensional vectors. The 65-dimensional Last.fm dataset is derived from the Last.fm music recommendation system [52], containing 292,385 vectors. GloVe-25 (50, 100, 200) dataset is a widely used pre-trained word embedding dataset containing 1,183,514 vectors [53]. GloVe-25 (50, 100, 200) indicates

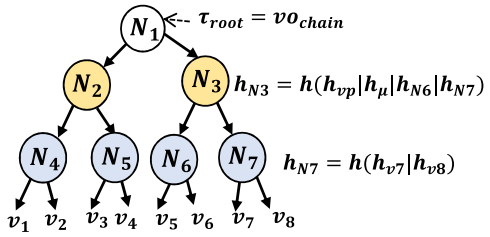


Fig. 8. An example of Merkle VP-tree.

that the vector dimensions in the dataset are 25 (50, 100, 200). We use the euclidean distance for SITT and the angular distance for Last.fm and GloVe.

Metrics. We evaluate ANNProof performance with the following metrics: (i) VO construction time, measured in terms of ANNProof server latency, (ii) VO and result verification time in terms of user latency, (iii) size of the VO, and (iv) overhead of the ADS construction protocol, measured in terms of ANNProof server latency and storage.

7.2. Compared verifiable query schemes

We evaluate four verifiable query schemes in experiments:

- **ImageProof**: This scheme facilitates verifiable SIFT-based similar image retrieval by implementing the Merkle randomized k-d tree and the Merkle inverted index with cuckoo filters as ADS [5].
- **Baseline**: This scheme employs the Merkle VP-tree as ADS to achieve verifiable K-ANNS queries. Due to the dense and highly correlated structure of the Merkle VP-tree, it is challenging to partition it into independent small segments for ADS optimization, making it unsuitable for sharding optimization methods [54].
- **ANNProof-U:** represents the performance of ANNProof using unsharded ADS.
- **ANNProof**: This scheme employs the Merkle HNSW node tree as ADS and incorporates the sharding techniques outlined in Definition 2 for optimization purposes.

Baseline implementation. We describe the ADS used in the Baseline: the Merkle VP-tree. Fig. 8 illustrates this tree, integrating both MHT and VP-tree structures. In a VP-tree, the leaf node contains a bucket of vector elements V_{leaf}, whereas the internal node contains the vantage point vp, the median distance value μ, and pointers to the left (right) child [54]. Thus, SP can build Merkle VP-tree in a bottom-to-top approach.

Digest of leaf node. Eq. (11) defines the digest of a Merkle VP-tree leaf node, where bs represents the bucket size, i.e., the number of vector elements in a leaf node.

$$h_{N_{leaf}} = h(v_1|v_2|...|v_{bs}) \tag{11}$$

Digest of internal node. The digest of a Merkle VP-tree internal node N_i is defined in Eq. (12). h_{vp} represents the hash digest of the node's vantage point, and h_{μ} is the median distance value digest. $h_{N_i^l}$ ($h_{N_i^r}$) is the digest of N_i's left (right) child.

$$h_{N_i} = h(h_{vp}|h_{\mu}|h_{N_i^l}|h_{N_i^r}) \tag{12}$$

7.3. Verifiable query performance of different schemes

Parameter settings. Regarding the HNSW parameters, we set: $M = 8, efConstruction = 100, efSearch = 50$. For the VP-tree, we set: $alphaLeft = 4.1, alphaRight = 2.3$. We set $sh_{size} = 10{,}000$ for sharding. We conduct experiments on the Sift-128 dataset, varying its size from

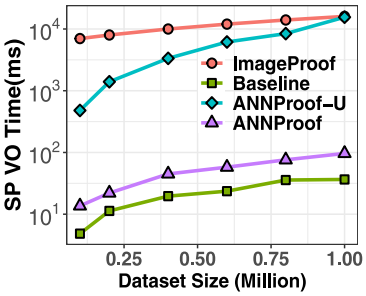


Fig. 9. VO construction time of SP.

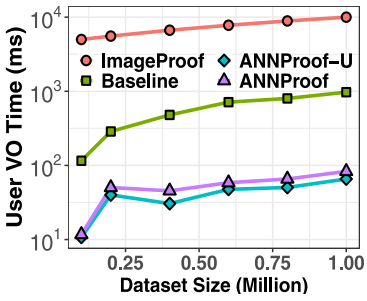


Fig. 10. VO verification time of users.

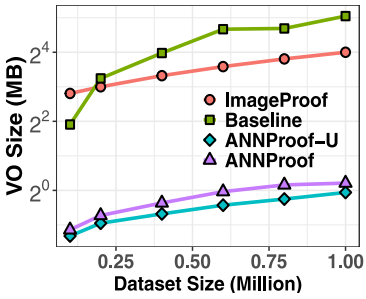


Fig. 11. VO size.

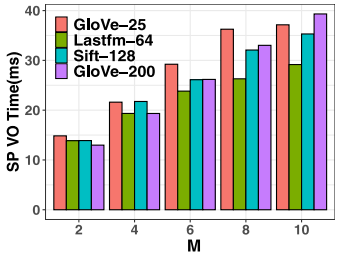


Fig. 12. VO construction time of SP with varying M.

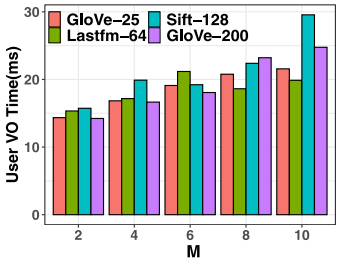


Fig. 13. VO and result verification time of users with varying M.

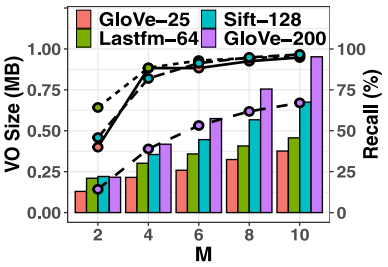


Fig. 14. VO size (bar chart) and query recall (line chart) with varying M.

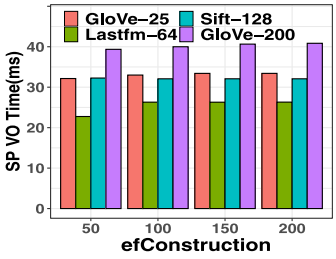


Fig. 15. VO construction time of SP with varying $efConstruction$.

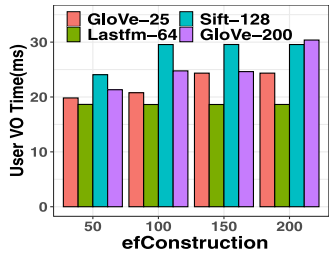


Fig. 16. VO and result verification time of users with varying $efConstruction$.

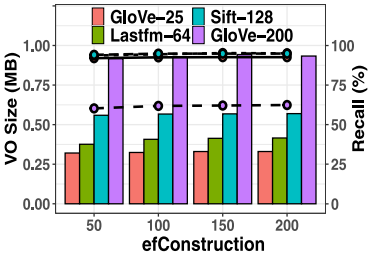


Fig. 17. VO size (bar chart) and query recall (line) with varying $efConstruction$.

0.1 ∼ 1 million. The recall for the dataset in the indexes consistently surpasses 95%.

VO construction time. Fig. 9 shows that ANNProof is 165 and 161 × faster than ImageProof and ANNProof-U, respectively. This is because ANNProof, leveraging sharding optimization, retains competitive performance by limiting ADS tree size to fixed complexity, dependent solely on sh_{size}. ImageProof uses the Merkle k-d tree and Merkle inverted index as ADS, along with the absence of the sharding technique in both ImageProof and ANNProof-U. As the dataset size and I_q size increases linearly, the complexity of I_q's VO generation grows exponentially, causing a significant elongation in VO generation time. In Fig. 9, ANNProof is 2 × slower than Baseline. Because, Baseline adopts the Merkle VP-tree as ADS, which possesses a significantly smaller tree size, thus achieving the fastest speed. Considering SPs have abundant

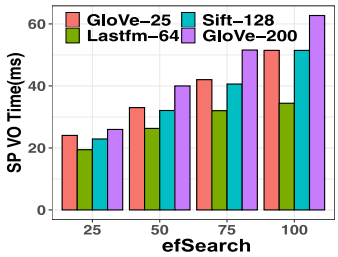


Fig. 18. VO construction time of SP with varying $efSearch$.

computational resources compared to users, the slightly longer VO generation time in SPs is acceptable.

Result verification time. While Fig. 10 shows that ANNProof is 1.1 × slower than ANNProof-U, it surpasses ANNProof-U by 161 × in the VO generation speed, according to Fig. 9. Furthermore, ANNProof maintains competitive performance in result verification, outpacing ImageProof and Baseline by 152 and 15 ×, respectively. Result verification time comprises two components: (i) I_q integrity verification time and (ii) local K-ANNS result search time. The latter is the dominant factor, and using the highly efficient HNSW search algorithm in the ANNProof significantly reduces the local search time compared to ImageProof and Baseline.

VO size. Fig. 11 shows that the VO size in ANNProof is only 3% to 7% of that in the Baseline and ImageProof. VO comprises two components: (i) VO_{sp}, which contains cryptographic proofs for validating I_q integrity, and (ii) I_q, where the latter occupies most of the VO size. Utilizing the HNSW search algorithm in ANNProof only requires delivering a minimal amount of data to users for the execution of a local search. Baseline exhibits the most considerable VO size due to its requirement to return substantial vectors in I_q. ImageProof necessitates returning users slightly fewer vectors than Baseline.

7.4. Verifiable query performance of ANNProof

We conduct experiments to investigate the impact of different parameters and datasets on the verifiable query efficiency in ANNProof. Specifically, we evaluate the performance using the Glove-25, Lastfm-64, Sift-128, and Glove-200 datasets, with a fixed dataset size of 0.2 million.

Impact of. M The parameter M denotes the number of connections between the nodes in H_{index}. Regarding the HNSW parameters, we set: $efConstruction = 100, efSearch = 50$, we vary M from 2 ∼ 10. Figs. 12 ∼ 14 illustrates the performance implications of varying M. Fig. 14 demonstrates that as M rises, there is a corresponding increase in query recall, resulting in higher query precision. However, Figs. 12, 13, and 14 show the increase in VO construction time, user verification time, and VO size with the elevation of M. With an increase in M, the connectivity of graph nodes in H_{index} also increases, resulting in the growth of the I_q size. Consequently, this leads to additional verification overheads for the same user query.

Impact of efConstruction. The $efConstruction$ parameter denotes the dynamic candidate list size when constructing H_{index}. We set: $M = 8, efSearch = 50$, we vary $efConstruction$ from 50 ∼ 200. Figs. 15 and 17 demonstrates the minimal impact of $efConstruction$ on recall, VO size, and VO generation time in SP. Fig. 16 illustrates an increase in user verification time for VO as $efConstruction$ rises and further increases as the dataset dimensionality grows.

Impact of efSearch. We set: $M = 8, efConstruction = 100$, we vary $efSearch = 50$ from 25 ∼ 100. Figs. 18 ∽ 20 shows the performance implications of varying $efSearch$, which denotes the dynamic candidate list size when searching H_{index}. With an increase in $efSearch$,

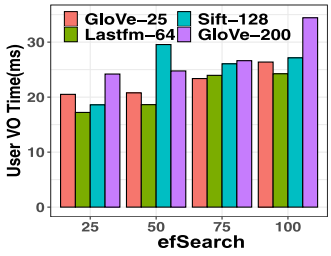


Fig. 19. VO and result verification time of users with varying $efSearch$.

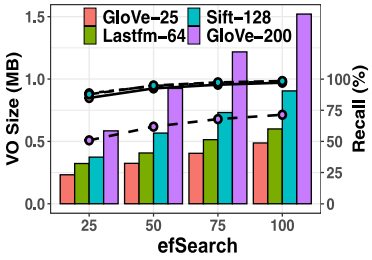


Fig. 20. VO size (bar chart) and query recall (line chart) with varying $efSearch$.

SP needs to search for more node neighbors in H_{index}, given the same user query. Therefore I_q size grows, leading to additional verification overheads and higher recall. Figs. 18, 19, and 20 illustrate the increase in VO construction time, user verification time, and VO size with the elevation of $efSearch$.

7.5. ADS construction protocol performance evaluation

ImageProof performs the worst across all metrics, showing an exponential performance gap compared to other schemes according to results in Section 7.3. The overheads of ANNProof-U and ANNProof on the ADS construction protocol are similar. Therefore, we compare Baseline and ANNProof in executing the protocol, highlighting why ANNProof is a superior solution for verifiable queries.

Parameter settings. The index parameters are the same as that in Section 7.3. We conduct experiments on the Sift-128 dataset, varying its size from 0.2 ∼ 1 million.

SP computation overhead. The time required for SP to execute the ADS construction protocol comprises (i) index building time and (ii) the time to construct the ADS based on the index. Fig. 21 shows that ANNProof is 2 × slower than Baseline in building index. Nevertheless, in Fig. 22, ANNProof's time percentage of ADS construction compared to index time is at most 2%, resulting in a low overhead for implementing the verifiable query scheme. Fig. 22 presents the ADS building time and its proportion relative to the index building time. A mere 1.6% of the indexing time is consumed by the ADS construction in ANNProof, attributable to the extensive HNSW indexing time.

SP storage overhead. Fig. 23 demonstrates that ANNProof's index size is twice as large as Baseline's. However, ANNProof achieves a smaller ADS size by employing a sharding optimization that is impossible with Baseline.

Conclusion. Although ANNProof's ADS construction protocol overhead is slightly elevated, it remains acceptable due to SPs' abundant computational resources compared to users.

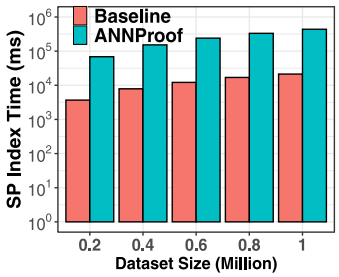


Fig. 21. Index building time.

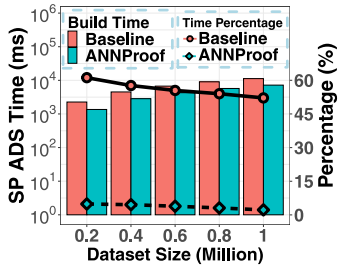


Fig. 22. ADS build time (bar) and its percentage of total index time (line).

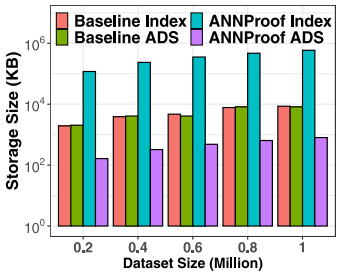


Fig. 23. Index size and ADS size.

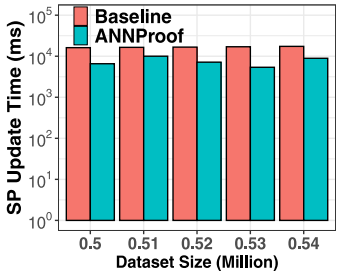


Fig. 24. Execution time of ADS update protocol with varying dataset size.

7.6. ADS update protocol performance evaluation

Protocol execution time. Fig. 24 shows that ANNProof is 2.5 × faster than Baseline in executing ADS update protocol. When the dataset size before the update is 510,000, the horizontal coordinate (x-axis) value of 0.51 corresponds to the protocol execution time when DP inserts 10,000 vectors into the dataset. The protocol execution time also consists of two components: (i) index update time and (ii) ADS update time.

Index update time. Fig. 25 shows that ANNProof is 2 × faster than Baseline in updating index. This is achieved through ANNProof's incremental vector insertion into the original H_{index}, obviating the necessity

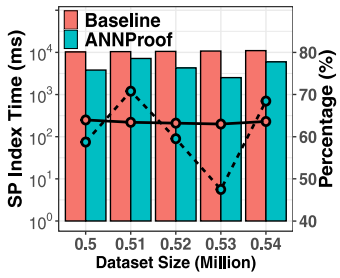


Fig. 25. Index update time (bar) and the percentage of ADS update time relative to it (line).

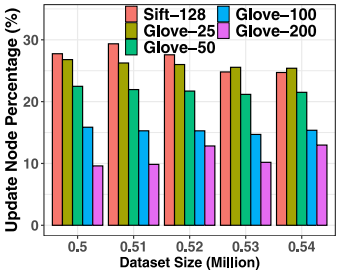


Fig. 26. Updated node percentage in the index updating process with varying datasets.

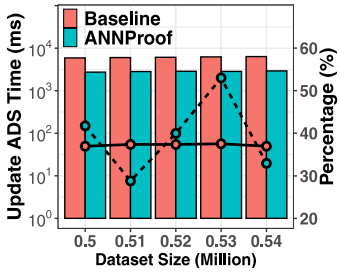


Fig. 27. ADS update time (bar chart) and its percentage of ADS update protocol's execution time (line chart).

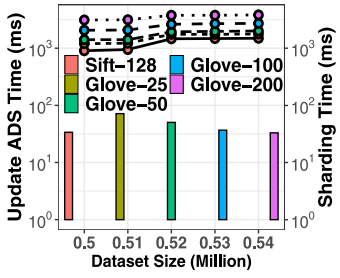


Fig. 28. Significantly extended update time of unsharded ADS (line chart) compared to sharded ADS (bar).

for the entire index reconstruction required by Baseline. When ANNProof updates H_{index}, it only updates a subset of nodes in H_{index} as shown in Fig. 26. For datasets such as Sift-128 and Glove-25 (50, 100, 200), the proportion of updated nodes in H_{index} ranges from 10% to 27%.

ADS update time. Fig. 27 shows that ANNProof is 2.2 × faster than Baseline in updating ADS. This discrepancy arises because Baseline must reconstruct the entire Merkle VP-tree to update ADS, while ANNProof employs the Merkle HNSW node tree with the sharding optimization. ANNProof only updates a subset of index nodes, resulting in the reconstruction of ADS only for the corresponding tree shards housing these nodes. The ADS update time of ANNProof accounts for at most 55% of ADS update protocol's execution time.

Sharding optimization. The line graph in Fig. 28 illustrates the ADS update time without sharding optimization. In this scenario, ANNProof must rebuild the Merkle tree for ADS update whenever DP inserts vectors into the dataset. The line graph aligns with the left y-axis, displaying how the unsharded ADS update time escalates with the growth of the original dataset. The Glove-200 dataset exhibits the longest update time. We evaluate five different datasets. When the pre-update dataset size is 510,000, an x-axis value of 0.51 indicates the ADS update time for inserting 10,000 vectors by DP.

The bar chart shows ADS update time with sharding optimization sh_{size} = 10,000, where each shard holds up to 10,000 leaf nodes. Under this setup, ADS updates in response to dataset size growth require only the construction of a new shard tree, leaving existing shards unchanged. Thus, the bar chart in Fig. 28 indicates that the update time is constant regardless of dataset size and varies only by dataset type. Comparing these two charts, we conclude that the sharding technique significantly reduces the ADS update time by 99%.

8. Related work

This section provides an overview of existing outsourced systems and conducts a comparative analysis between ANNProof and state-of-the-art systems, highlighting the advantages of ANNProof.

8.1. Conventional outsourced query system (OQS)

Verifiable queries can be achieved through the use of authenticated data structure (ADS). Prior studies, such as SVLSQ [55] and PA^2 [33], employ cryptographic primitives to design ADS, enabling verifiable query semantics, such as set operations and Skyline queries. MHT is a widely employed cryptographic primitive in the design of ADS. For instance, the Merkle-B tree (MB-tree) [27] provides a verifiable range query method, while VN-Auth [31] combines MTH and VoR-Tree for verifiable spatial nearest neighbor (NN) queries.

Verifiable query semantics supported by MHT include integrity verification, efficient comparison, and proof of existence [34,56]. MHT-based ADS exhibit low computational overhead due to the efficiency of Merkle hash operations. Verifiable query semantics that employs complex primitives and protocols, such as accumulators [16] and interactive proof protocols [55], experience higher computational overhead. Unfortunately, DP's susceptibility to attacks may lead to outsourcing tampered data to SPs, causing users to misperceive honest SPs as malicious.

8.2. Public blockchain-based OQS

Blockchain. Ethereum is the pioneering public blockchain to incorporate smart contracts, allowing users to read and write data from the blockchain [57]. Permissioned blockchain, e.g., Fabric [40], and FISCO BCOS [58] achieve better performance and also meet the security requirements in commercial scenarios [32,59,60].

Blockchain technology provides a promising solution for preventing tampered data outsourcing from DP through its secure data storage capabilities. For instance, GEM^2 [29] uses smart contracts to form a tamper-evident ADS and integrates a gas-efficient Merkle merge tree to decrease ADS size. The SGX-based query framework [21] establishes a protocol between trusted hardware and smart contracts for ADS construction. iQuery [20] applies a game-theory-based smart contract to confirm the query results' correctness. Some approaches involve miners in the construction of ADS within the blockchain. For example, in vChain [16], miners construct accumulator-based ADS and incorporate them into each block, while VQL [22] necessitates miners to verify the data layer's consistency, significantly consuming computational resources.

Table 2
Comparison of outsourced query systems (OQS)

Catalog	Representative	C1			C2
		Query Semantic	Verification Overhead	Economic Cost	Tamper-evident Dataset
Conventional OQS	SVLSQ [55], PA^2 [33], MB-tree [27], VN-Auth [31], ImageProof [5]	SVLSQ: Skyline, PA^2: Set,MB-tree: Range, VN-Auth:NN, ImageProof: Topk images	MB-tree, VN-Auth, ImageProof: Medium, Others: High	Small	×
Public Blockchain Based OQS	GEM^2 [29], vChain [16], VQL [22], iQuery [20], SGX-based query framework [21]	iQuery: Arbitrary, Others: Boolean range and time-window	vChain: High, iQuery: Small, Others: Medium	iQuery: Twice the expenses, GEM^2:High Others: Small	✓
Permissioned Blockchain Based OQS	ANNProof	K-approximate nearest neighbor search	Medium	Small	✓

8.3. Comparison with state-of-the-art systems

Table 2 compares conventional OQS, public blockchain-based OQS, and ANNProof. Among these state-of-the-art systems, ANNProof represents the first endeavor to construct a verifiable, outsourced K-ANN search system on blockchain. To address C1, ANNProof incorporates optimized designs in query semantics, verification overhead, and economic cost.

Query Semantic. While ImageProof [5] is exclusively suited for similar image retrieval, ANNProof broadens **versatility** with its K-ANNS semantics, accommodating diverse applications from service recommendations to anti-money laundering [7–9]. Authenticated query semantics in vChain, GEM^2, VQL, and the SGX-based query framework are limited to boolean range and time-window queries. iQuery supports arbitrary query semantics. However, it necessitates *two* SPs to deliver results, increasing economic costs.

Verification Overhead. Compared to ImageProof, ANNProof significantly diminishes the user verification overhead by using an efficient search index. Unlike GEM^2, ANNProof stores the complete ADS off-chain, retaining only the ADS root on the blockchain. By using ADS construction smart contract and endorsement strategies, ANNProof significantly reduces the information stored on the blockchain while maintaining trustworthiness, striking a balance between security and efficiency.

Compared to vChain, ANNProof employs MHT-based ADS, resulting in lower computational overhead. Additionally, accumulator primitives' calculation overhead is 4 × greater than MHT [16]. Besides, VQL and the SGX-based query framework also exhibit lower ADS calculation overheads due to the utilization of MHT.

Economic Cost. iQuery incurs twice the expenses in verification due to the game-theory-based incentive mechanism. In contrast, ANNProof only requires users to pay for a single outsourced query, as users can verify results with VO, resulting in reduced economic costs.

GEM^2 incurs substantial **gas fees** when storing the complete ADS on the blockchain. Although the trustworthiness of the ADS construction process is ensured, storing information on the public blockchain is **costly**, leading to considerable overheads in outsourced query fees [61,62]. Besides, within EVM, the gas fees for utilizing the Merkle hash primitive is **much cheaper** than the accumulator primitive [35]. Consequently, vChain modifies the block data structure and directs miners to compute accumulators for ADS, eliminating gas fees required for using EVM. However, this design is **inapplicable** to public blockchain systems typically employing the EVM execution environment [59]. In contrast, ANNProof offers a verifiable query scheme applicable to general permissioned blockchains without modifying the underlying infrastructure, enhancing its **generality**. This enhancement enables ANNProof to deliver a highly cost-effective authenticated K-ANNS query framework.

Tamper-evident Datasets. By leveraging blockchain smart contracts, ANNProof ensures DP provides incorruptible data, effectively addressing C2 compared to OQS approaches.

9. Conclusion and future work

This paper presents the ANNProof framework, the first exploration of verifiable K-ANNS on the blockchain. HNSW is the optimal solution for verifiable outsourcing, which provides a smaller partial index size, shorter search time, and determined search paths, enabling efficient local result search and query verification for users. We propose two novel ADSs to ensure the integrity of (i) the vector query algorithm and (ii) the vector-identifier relationship, optimizing ADS using sharding techniques. Security analysis formally confirms that ANNProof ensures query results' soundness, completeness, and freshness. Extensive experiments demonstrate its superior performance over state-of-the-art systems.

In the future, we will explore blockchain-based payment methods for outsourced queries and aim to design a settlement approach that is fair, secure, and gas-efficient.

CRediT authorship contribution statement

Lingling Lu: Conceptualization, Methodology, Software. **Zhenyu Wen:** Writing – original draft, Writing – review & editing. **Ye Yuan:** Investigation, Supervision. **Qinming He:** Funding acquisition, Supervision. **Jianhai Chen:** Formal analysis, Validation. **Zhenguang Liu:** Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This paper is financially supported by the National Key R&D Program of China (2021YFB2700500, 2021YFB2700501) and Key R&D Program of Zhejiang Province (2022C01086). Zhenyu Wen is supported by the China Postdoctoral Science Foundation under Grant 2023M743403 and the Zhejiang Provincial Natural Science Foundation of Major Program (Youth Original Project) under Grant LDQ24F020001. Ye Yuan is supported by the National Key Research and Development Program of China (Grant No. 2022YFB2702100) and the NSFC (Grant Nos. 61932004, 62225203, U21A20516). We thank all of the anonymous reviewers who spent their own time reviewing and giving suggestions. We thank Hanning Ruan (Transwarp) for improving paper writing, and Jianting He (BlockSec) for helping to build the blockchain experiments.

References

[1] Google, Google cloud: Cloud computing services, 2023, https://cloud.google.com/ (Accessed 26 June 2023).
[2] Amazon, Amazon web services (AWS), 2023, https://aws.amazon.com/ (Accessed 26 June 2023).
[3] Microsoft, Microsoft azure: Cloud computing services, 2023, https://azure.microsoft.com/ (Accessed 26 June 2023).
[4] B. Naidan, L. Boytsov, E. Nyberg, Permutation search methods are efficient, yet faster search is possible, in: Proceedings of the 41st International Conference on Very Large Data Bases, VLDB, vol. 8, VLDB Endowment, 2015, pp. 1618–1629.
[5] S. Guo, J. Xu, C. Zhang, C. Xu, T. Xiang, ImageProof: Enabling authentication for large-scale image retrieval, in: Proceedings of the 35th International Conference on Data Engineering, ICDE, IEEE, 2019, pp. 1070–1081.
[6] H. Pang, K. Mouratidis, Authenticating the query results of text search engines, in: Proceedings of the 34th International Conference on Very Large Data Bases, VLDB, vol. 1, VLDB Endowment, 2008, pp. 126–137.
[7] M. Aumüller, E. Bernhardsson, A. Faithfull, ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms, Inform. Syst. (IS) 87 (2020) 101374.
[8] Y. Xia, Y. Cao, S. Hu, T. Liu, L. Lu, Deep intention-aware network for click-through rate prediction, in: Proceedings of the 31st International World Wide Web Conferences, WWW, ACM, 2022.
[9] Y. Yuan, D. Ma, Z. Wen, Z. Zhang, G. Wang, Subgraph matching over graph federation, in: Proceedings of the 47th International Conference on Very Large Data Bases (VLDB 2019), vol. 15, (3) VLDB Endowment, 2021, pp. 437–450.
[10] S. Hu, Z. Zhang, B. Luo, S. Lu, B. He, L. Liu, BERT4eth: A pre-trained transformer for ethereum fraud detection, in: Proceedings of the 32nd International World Wide Web Conferences, WWW, 2023, pp. 2189–2197.
[11] Z. Huang, Y. Huang, P. Qian, J. Chen, Q. He, Demystifying bitcoin address behavior via graph neural networks, in: Proceedings of the 39th International Conference on Data Engineering, ICDE, 2023.
[12] Demyst, Demyst: The external data platform, 2023, https://demyst.com/ (Accessed: 26 June 2023).
[13] PeerIQ, Peeriq: Technology for the next wave of alternative lending, 2023, https://www.peeriq.com/ (Accessed: 26 June 2023).
[14] K. Ren, Y. Guo, J. Li, X. Jia, C. Wang, Y. Zhou, S. Wang, N. Cao, F. Li, Hybridx: New hybrid index for volume-hiding range queries in data outsourcing services, in: Proceedings of the 40th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2020, pp. 23–33.
[15] M. Zhang, Z. Xie, C. Yue, Z. Zhong, Spitz: a verifiable database system, Proceedings of the 46th International Conference on Very Large Data Bases (VLDB), vol. 13 (2020) 3449–3460.
[16] C. Xu, C. Zhang, J. Xu, Vchain: Enabling verifiable boolean range queries over blockchain databases, in: Proceedings of the 45th International Conference on Management of Data, SIGMOD, 2019, pp. 141–158.
[17] S. Wu, L. Wu, Y. Zhou, R. Li, Z. Wang, X. Luo, C. Wang, K. Ren, Time-travel investigation: Toward building a scalable attack detection framework on ethereum, ACM Trans. Softw. Eng. Methodol. (TOSEM) 31 (3) (2022) 1–33.
[18] L. Lu, Verifiable non-metric space library (v-NMSLIB), 2023, https://github.com/lulinglingcufe/vnmslib/ (Accessed: 26 May 2023).
[19] A. Sharma, F.M. Schuhknecht, D. Agrawal, J. Dittrich, Blurring the lines between blockchains and database systems: the case of hyperledger fabric, in: Proceedings of the 45th International Conference on Management of Data, SIGMOD, 2019, pp. 105–122.
[20] L. Lu, Z. Wen, Y. Yuan, B. Dai, P. Qian, C. Lin, Q. He, Z. Liu, J. Chen, R. Ranjan, Iquery: A trustworthy and scalable blockchain analytics platform, IEEE Trans. Depend. Secure Comput. (TDSC) (2023).
[21] C. Cai, L. Xu, A. Zhou, C. Wang, Toward a secure, rich, and fair query service for light clients on public blockchains, IEEE Trans. Depend. Secure Comput. (TDSC) (2021).
[22] H. Wu, Z. Peng, S. Guo, Y. Yang, B. Xiao, VQL: efficient and verifiable cloud query services for blockchain systems, IEEE Trans. Parallel Distrib. Syst. (TPDS) 33 (6) (2021) 1393–1406.
[23] Q. Zhang, Y. He, R. Lai, Z. Hou, G. Zhao, A survey on the efficiency, reliability, and security of data query in blockchain systems, Future Gener. Comput. Syst. 145 (2023) 303–320.
[24] S. Zhang, J. He, W. Liang, K. Li, MMDS: A secure and verifiable multimedia data search scheme for cloud-assisted edge computing, Future Gener. Comput. Syst. (2023).
[25] Q. Wang, F. Zhou, J. Xu, Z. Xu, Efficient verifiable databases with additional insertion and deletion operations in cloud computing, Future Gener. Comput. Syst. 115 (2021) 553–567.
[26] Y. Liu, J. Yu, M. Yang, W. Hou, H. Wang, Towards fully verifiable forward secure privacy preserving keyword search for IoT outsourced data, Future Gener. Comput. Syst. 128 (2022) 178–191.
[27] F. Li, M. Hadjieleftheriou, G. Kollios, L. Reyzin, Dynamic authenticated index structures for outsourced databases, in: Proceedings of the ACM International Conference on Management of Data, SIGMOD, 2006, pp. 121–132.
[28] R.C. Merkle, A digital signature based on a conventional encryption function, in: Proceedings of the 6th Annual International Conference on the Theory and Applications of Cryptographic Techniques, EUROCRYPT, Springer, 1987, pp. 369–378.
[29] C. Zhang, C. Xu, J. Xu, Y. Tang, B. Choi, Gem^2-tree: A gas-efficient structure for authenticated range queries in blockchain, in: Proceedings of the 35th International Conference on Data Engineering, ICDE, IEEE, 2019, pp. 842–853.
[30] M.L. Yiu, Y. Lin, K. Mouratidis, Efficient verification of shortest path search via authenticated hints, in: Proceedings of the 26th International Conference on Data Engineering, ICDE, IEEE, 2010, pp. 237–248.
[31] L. Hu, W.-S. Ku, S. Bakiras, C. Shahabi, Spatial query integrity with voronoi neighbors, IEEE Trans. Knowl Data Eng. (TKDE) 25 (4) (2011) 863–876.
[32] U. Demirbaga, G.S. Aujla, MapChain: A blockchain-based verifiable healthcare service management in IoT-based big data ecosystem, IEEE Trans. Netw. Serv. Manage. (TNSM) (2022).
[33] C. Xu, Q. Chen, H. Hu, J. Xu, X. Hei, Authenticating aggregate queries over set-valued data with confidentiality, IEEE Trans. Knowl. Data Eng. 30 (4) (2017) 630–644.
[34] Y. Zhang, D. Genkin, J. Katz, D. Papadopoulos, C. Papamanthou, vSQL: Verifying arbitrary SQL queries over dynamic outsourced databases, in: Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P), IEEE, 2017, pp. 863–880.
[35] P. Qian, J. He, L. Lu, S. Wu, Z. Lu, L. Wu, Y. Zhou, Q. He, Demystifying random number in ethereum smart contract: Taxonomy, vulnerability identification, and attack detection, IEEE Trans. Softw. Eng. (TSE) (2023).
[36] Y.A. Malkov, D.A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, IEEE Trans. Pattern Anal. Mach. Intell. (TPAMI) 42 (4) (2018) 824–836.
[37] L. Boytsov, D. Novak, Y. Malkov, E. Nyberg, Off the beaten path: Let's replace term-based retrieval with k-nn search, in: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, CIKM, 2016, pp. 1099–1108.
[38] J.K. Uhlmann, Satisfying general proximity/similarity queries with metric trees, Inform. Process. Lett. 40 (4) (1991) 175–179.
[39] P.N. Yianilos, Data structures and algorithms for nearest neighbor, in: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, vol. 66, 1993, p. 311.
[40] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the 13th European Conference on Computer Systems Conference, EuroSys, ACM, 2018, pp. 1–15.
[41] P. Zhang, Y. Wang, G.S. Aujla, A. Jindal, Y.D. Al-Otaibi, A blockchain-based authentication scheme and secure architecture for IoT-enabled maritime transportation systems, IEEE Trans. Intell. Transport. Syst. (TITS) 24 (2) (2022) 2322–2331.
[42] A. Jindal, G.S. Aujla, N. Kumar, M. Villari, GUARDIAN: Blockchain-based secure demand response management in smart grid system, IEEE Trans. Serv. Comput. (TSC) 13 (4) (2019) 613–624.
[43] L. Lao, X. Dai, B. Xiao, S. Guo, G-PBFT: a location-based and scalable consensus protocol for IOT-blockchain applications, in: Proceedings of the 34th International Parallel and Distributed Processing Symposium, IPDPS, IEEE, 2020, pp. 664–673.
[44] H. Dang, T.T.A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, B.C. Ooi, Towards scaling blockchain systems via sharding, in: Proceedings of the 45th International Conference on Management of Data, SIGMOD, 2019, pp. 123–140.
[45] J. Behl, T. Distler, R. Kapitza, Hybrids on steroids: SGX-based high performance BFT, in: Proceedings of the 12th European Conference on Computer Systems, EuroSys, ACM, 2017, pp. 222–237.
[46] A. Bessani, J. Sousa, M. Vukolić, A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform, in: Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, SERIAL, 2017, pp. 1–2.

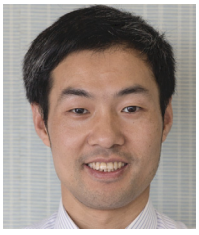
[47] S. Liu, P. Viotti, C. Cachin, V. Quéma, M. Vukolic, XFT: Practical fault tolerance beyond crashes, in: Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementations, OSDI, 2016, pp. 485–500.
[48] T.L.F. Project, Hyperledger fabric, 2023, https://github.com/hyperledger/fabric (Accessed 26 May 2023).
[49] L. Boytsov, Non-metric space library (NMSLIB), 2023, https://github.com/nmslib/nmslib/ (Accessed 26 May 2023).
[50] G. Bertoni, J. Daemen, M. Peeters, G. Van Assche, Keccak, in: Proceedings of the 32nd European Cryptology Conference, EUROCRYPT, Springer, 2013, pp. 313–314.
[51] L. Amsaleg, Datasets for approximate nearest neighbor search, 2023, http://corpus-texmex.irisa.fr/ (Accessed 26 June 2023).
[52] B. Frederickson, Approximate nearest neighbours for recommender systems, 2023, http://www.benfrederickson.com/approximate-nearest-neighbours-for-recommender-systems/ (Accessed 26 June 2023).
[53] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: Proceedings of the 19th Conference on Empirical Methods in Natural Language Processing, EMNLP, 2014, pp. 1532–1543.
[54] T.-c. Chiueh, Content-based image indexing, in: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, vol. 94, Citeseer, 1994, pp. 582–593.
[55] Z. Wang, X. Ding, H. Jin, P. Zhou, Efficient Secure and Verifiable Location-Based Skyline Queries over Encrypted Data, 15, (9) VLDB Endowment, 2022, pp. 1822–1834,
[56] R.S. Wahby, S. Setty, M. Howald, Z. Ren, A.J. Blumberg, M. Walfish, Efficient RAM and control flow in verifiable outsourced computation, in: Proceedings of the 22nd Network and Distributed System Security Symposium, NDSS, Internet Society, 2015.
[57] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) (2014) 1–32.
[58] F.B. Community, FISCO-BCOS, 2021, https://github.com/FISCO-BCOS (Accessed 20 July 2021).
[59] M. Li, Y. Wang, S. Ma, C. Liu, D. Huo, Y. Wang, Z. Xu, Auto-tuning with reinforcement learning for permissioned blockchain systems, 16, (5) VLDB Endowment, 2023, pp. 1000–1012,
[60] L. Yuan, Q. He, S. Tan, B. Li, J. Yu, F. Chen, Y. Yang, CoopEdge+: Enabling decentralized, secure and cooperative multi-access edge computing based on blockchain, IEEE Trans. Parallel Distrib. Syst. (TPDS) 34 (3) (2023) 894–908.
[61] Z. Sui, J.K. Liu, J. Yu, X. Qin, Monet: A fast payment channel network for scriptless cryptocurrency monero, in: Proceedings of the 42nd International Conference on Distributed Computing Systems, ICDCS, IEEE, 2022, pp. 280–290.
[62] R. Han, Z. Sui, J. Yu, J. Liu, S. Chen, Fact and fiction: Challenging the honest majority assumption of permissionless blockchains, in: Proceedings of the ACM Conference on Computer and Communications Security, CCS, ACM, 2021, pp. 817–831.

Lingling Lu is currently pursuing the Ph.D. degree with the College of Computer Science and Technology, Zhejiang University, Hangzhou, China. She engages in blockchain security and optimization research work. Her research interests include blockchain, database managements and big data processing.

Zhenyu Wen (Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from Newcastle University, Newcastle Upon Tyne, U.K., in 2011 and 2016, respectively. He is currently a Professor with the Institute of Cyberspace Security and college of Information Engineering, Zhejiang University of Technology, China. His current research interests include IoT, crowd sources, AI system, and cloud computing. For his contributions to the area of scalable data management for the internet-of-things, he was awarded the IEEE TCSC Award for Excellence in Scalable Computing (Early Career Researchers) in 2020.

Ye Yuan received his BS, MS and Ph.D. degrees in Computer Science from Northeastern University, China in 2004, 2007 and 2011, respectively. He is now a Professor at the College of Information Science and Engineering at Beijing Institute of Technology. His research interests include graph databases, probabilistic databases, data privacy-preserving and cloud computing.

Qinming He (Member, IEEE) received the BS, MS, and Ph.D. degrees in computer science from Zhejiang University, Hangzhou, P. R. China, in 1985, 1988, and 2000, respectively. He is currently a professor with the College of Computer Science & Technology, Zhejiang University. His research interests include data mining and blockchain system security.



Jianhai Chen (Member, IEEE) received the MS and Ph.D. degrees in computer science and technology from Zhejiang University (ZJU), Hangzhou, China. He is currently an associate professor of the College of Computer Science and Technology, ZJU. His research interests include blockchain system security, cloud computing scheduling algorithms and game theory.

Zhenguang Liu is currently a professor of Zhejiang University. He had been a research fellow in National University of Singapore, and A*STAR (Agency for Science, Technology and Research, Singapore) for three years. He respectively received his Ph.D. and B.E. degrees from Zhejiang University and Shandong University, China, in 2010 and 2015. His research interests include blockchain security and multimedia data analysis.